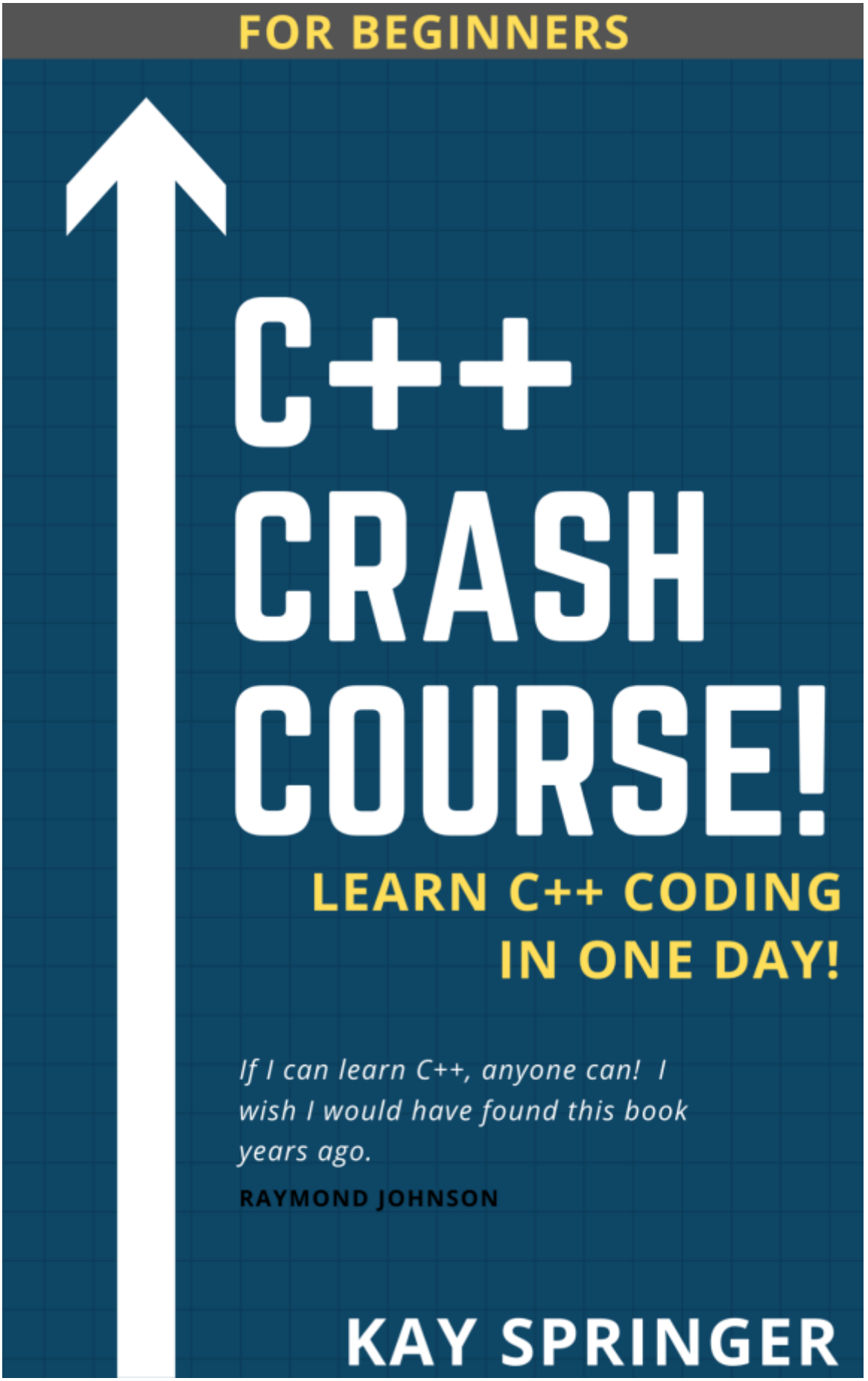

FOR BEGINNERS
C++
CRASH
COURSE!
LEARN C++ CODING
IN ONE DAY!
If I can learn C++, anyone can! I wish I would have found this book years ago.
RAYMOND JOHNSON
KAY SPRINGER

INSTALLING

BUILD OUR FIRST CONSOLE PROGRAM

STRUCTURE OF OUR PROGRAM

IOSTREAM

THE MAIN

DATA TYPES

VARIABLES

CONCATENATION - STRING LITERALS

PRINTING STRINGS & INTEGERS

MULTI-LINE STRINGS

PRACTICE ACTIVITIES

ARITHMETIC OPERATORS

COMPARISON OPERATORS

LOGICAL OPERATORS

PRACTICE ACTIVITIES

OPERATORS & BOOLEANS

NAMESPACE

THE STRING OBJECT

USER INPUT

PRACTICE ACTIVITIES

MATH

ARRAYS

PRACTICE ACTIVITIES

PRACTICE ACTIVITY

IF-ELSE STATEMENTS

FOR LOOPS

WHILE LOOP

BREAK / CONTINUE

FUN WITH FUNCTIONS

OOP CLASSES

CLASS METHODS

CONSTRUCTORS

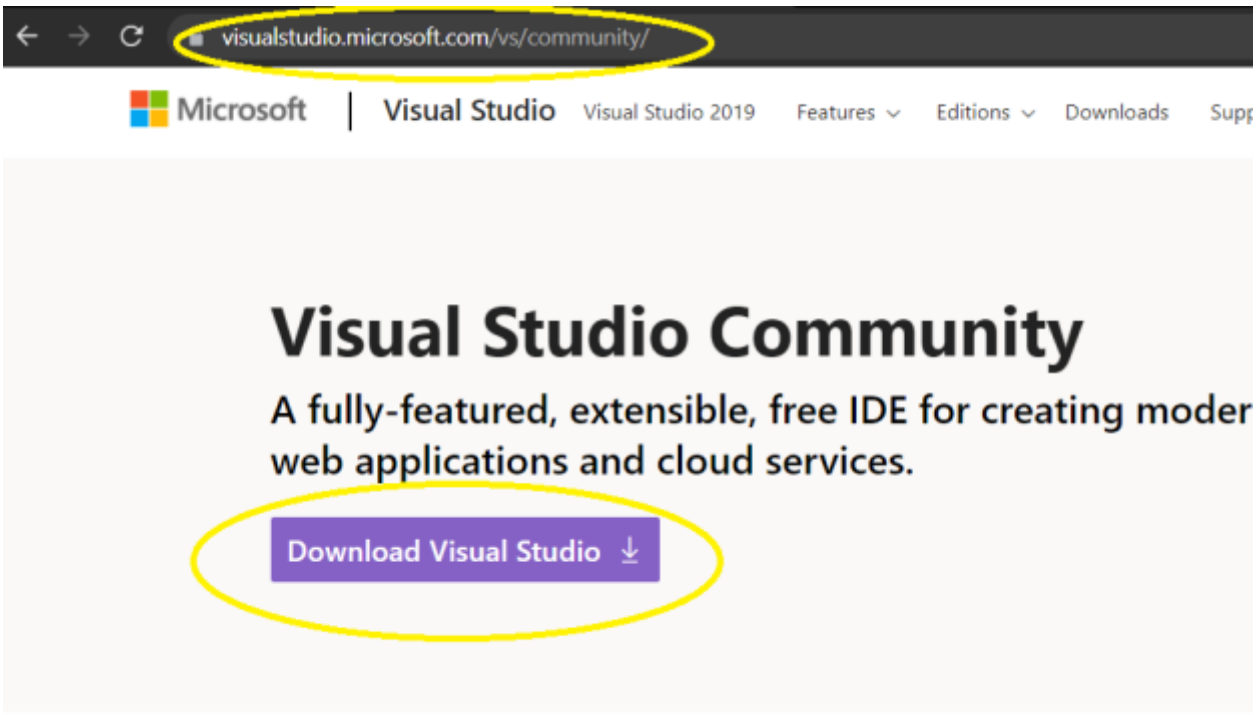
FILES

So you want to learn C++? C++ is a **powerful general-purpose programming** It can be used to develop operating systems, browsers, games, and so much more. Popular games developed using C++ include Assassins Creed, Farcry and Family Guy to name a few.

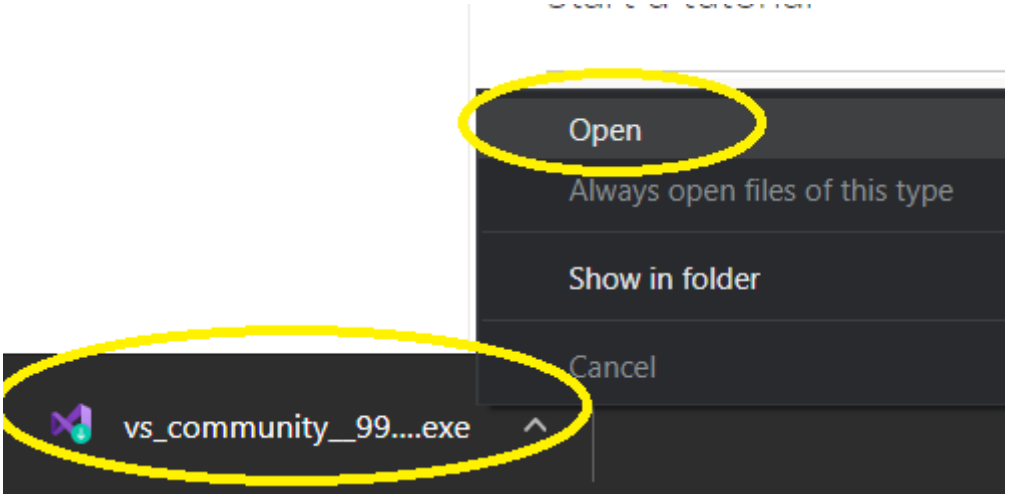
This book will start off with super basics and will guide you step-by-step to becoming a fully capable C++ programmer. So let's skip all the fancy intros and start learning some C++!

INSTALLING

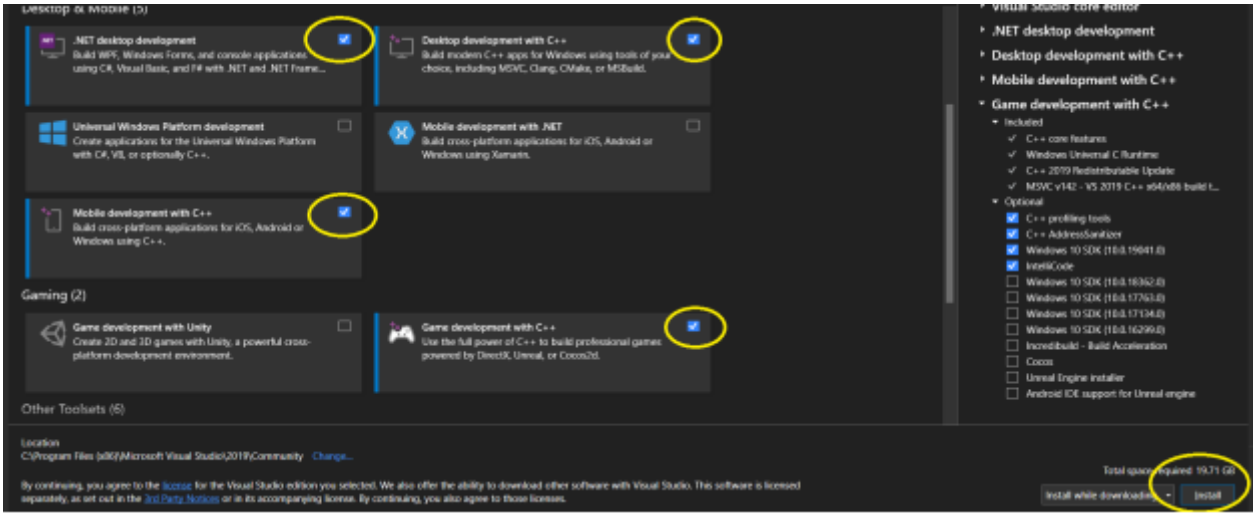
We will start by installing a free Integrated Development Environment (IDE) called Visual Studio Community by going to https://visualstudio.microsoft.com/vs/community/



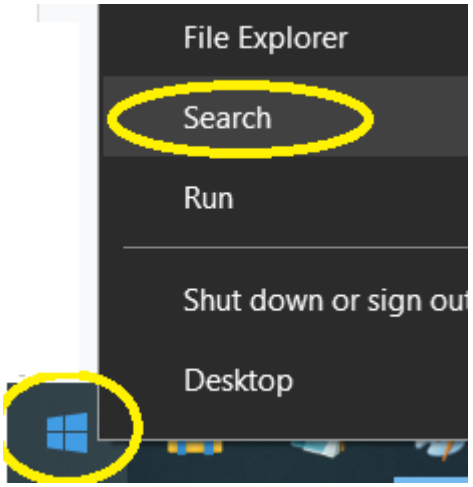
If using Chrome, your download will show on the bottom left corner. Right click the file and select 'Open' or 'Open When Done'

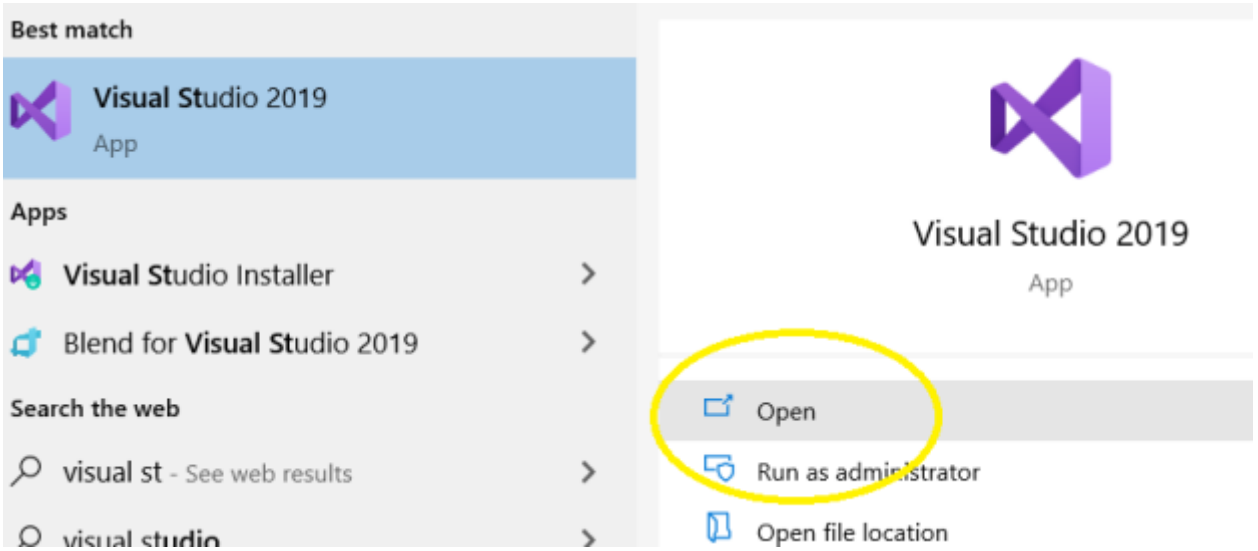


Run through the installation process, then you should be presented with a screen asking what you want to install with your Visual Studio IDE. For now we need to choose .NET Desktop, Desktop Development with C++, Mobile Development with C++ and Game Development with C++. Choose these then click 'install'

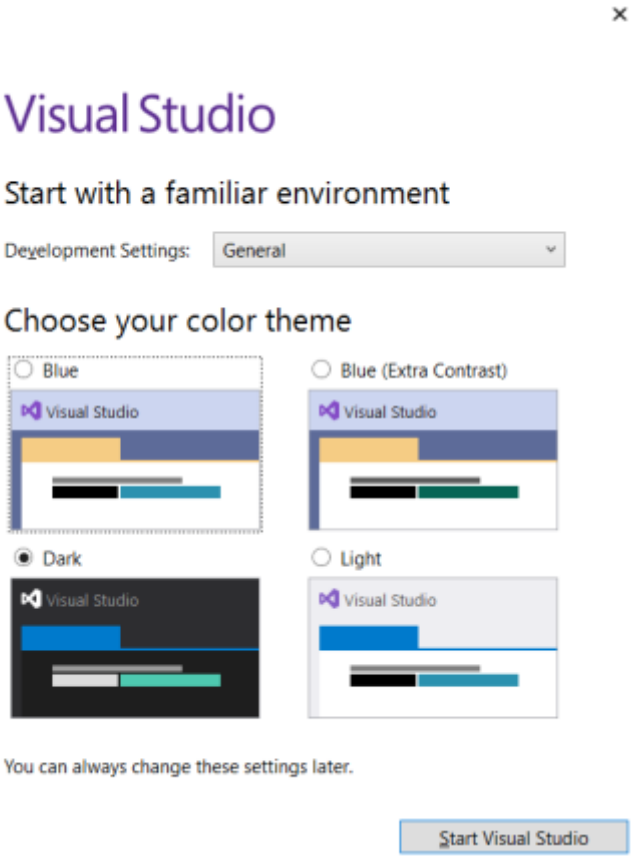


After installation is complete, your IDE will reboot your computer. Now let's go find our IDE by clicking on the bottom left hand Windows icon and search for Visual Studio.

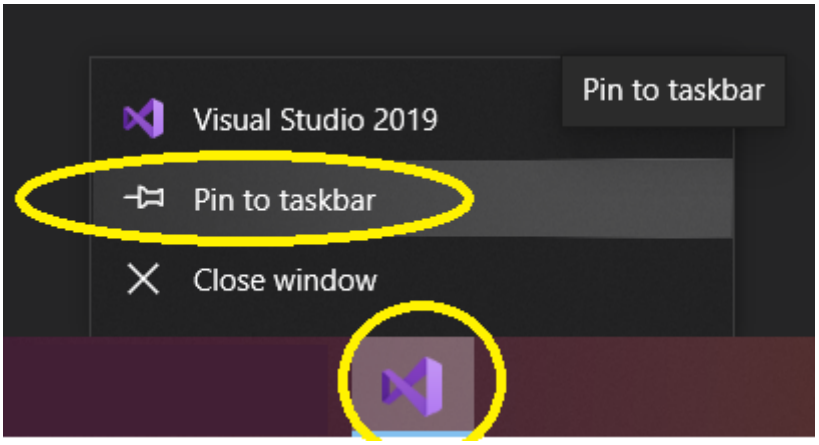




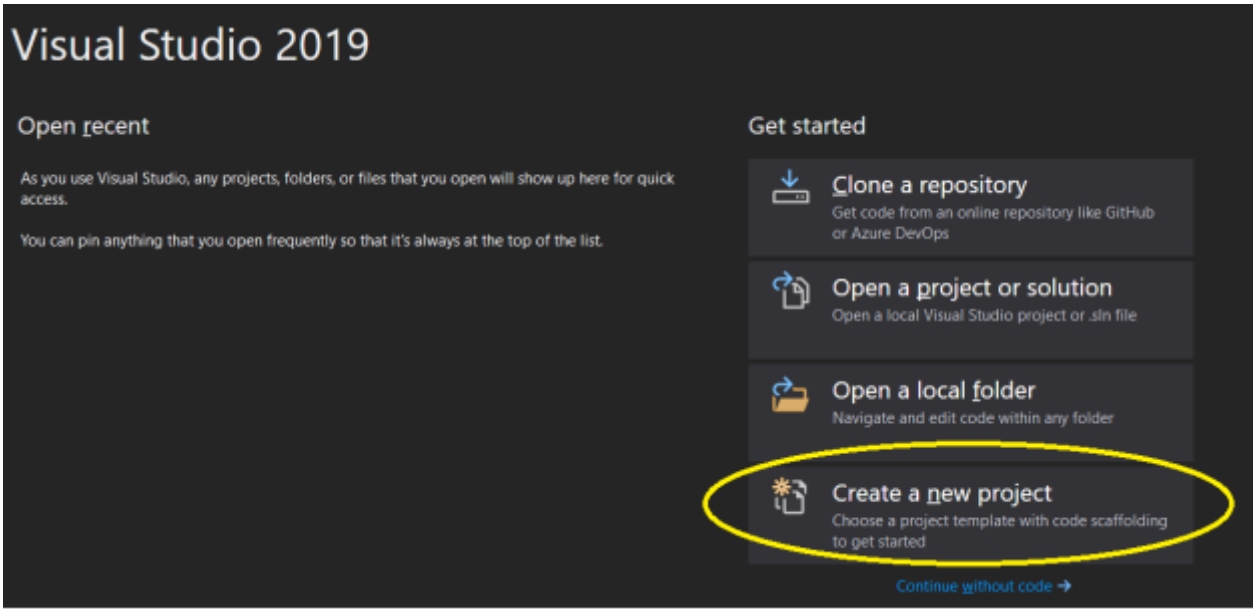
As your IDE opens it will ask you to sign in, create an account or skip. Skip this option unless you want to create an account or sign in. The next prompt will ask what color you want your environment. Dark settings show a black background with light text, while the other selections are a white background with dark text. I will be choosing the dark mode.



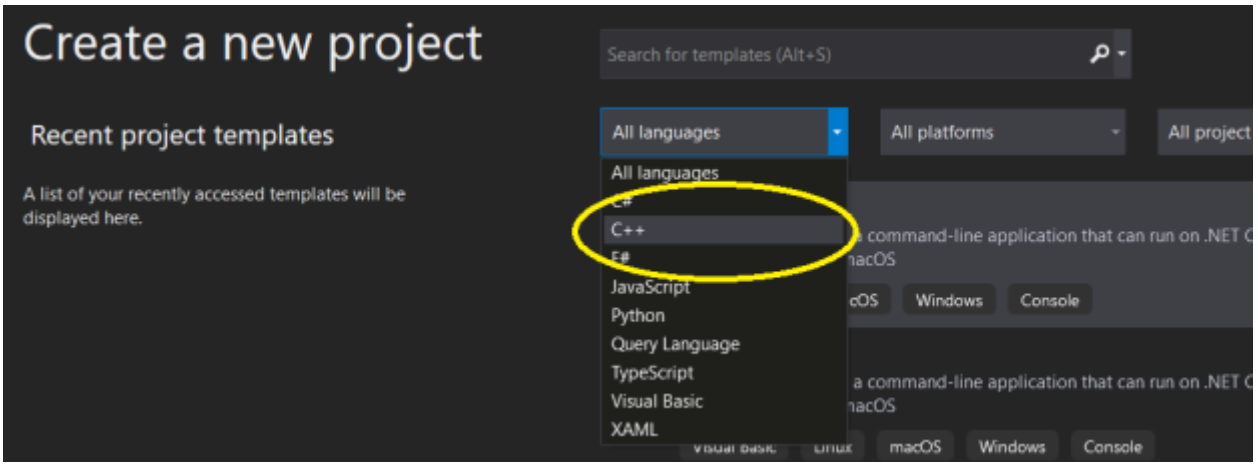
After your IDE has launched, the Visual Studio icon will show on your taskbar. Right click and choose 'pin to taskbar'. This will make your life easier so you do not have to go search and locate your IDE every time you want to code!

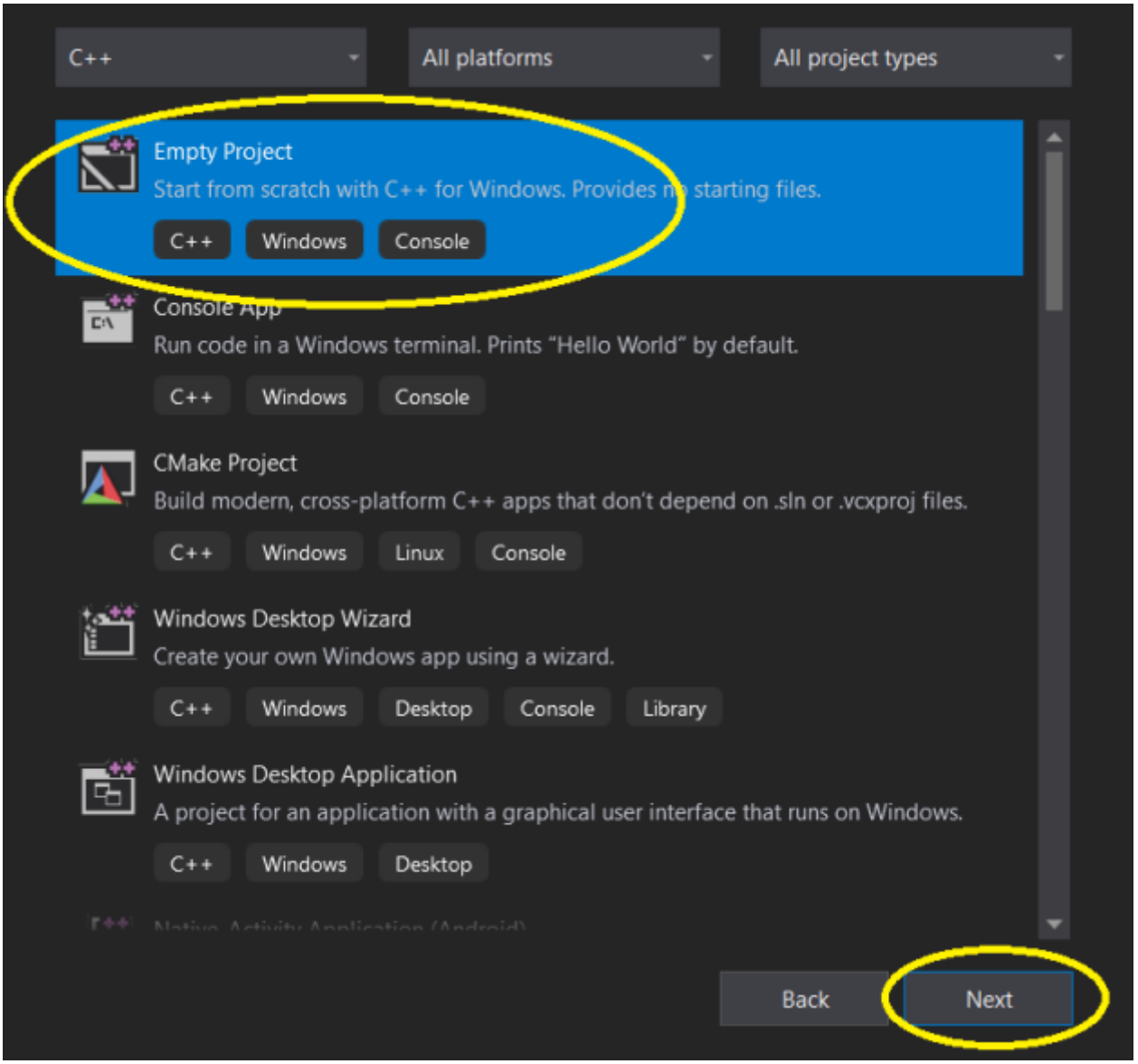


Whew! We are finally finished with our IDE installation! Now you will be presented with the following screen where we need to start a new project:

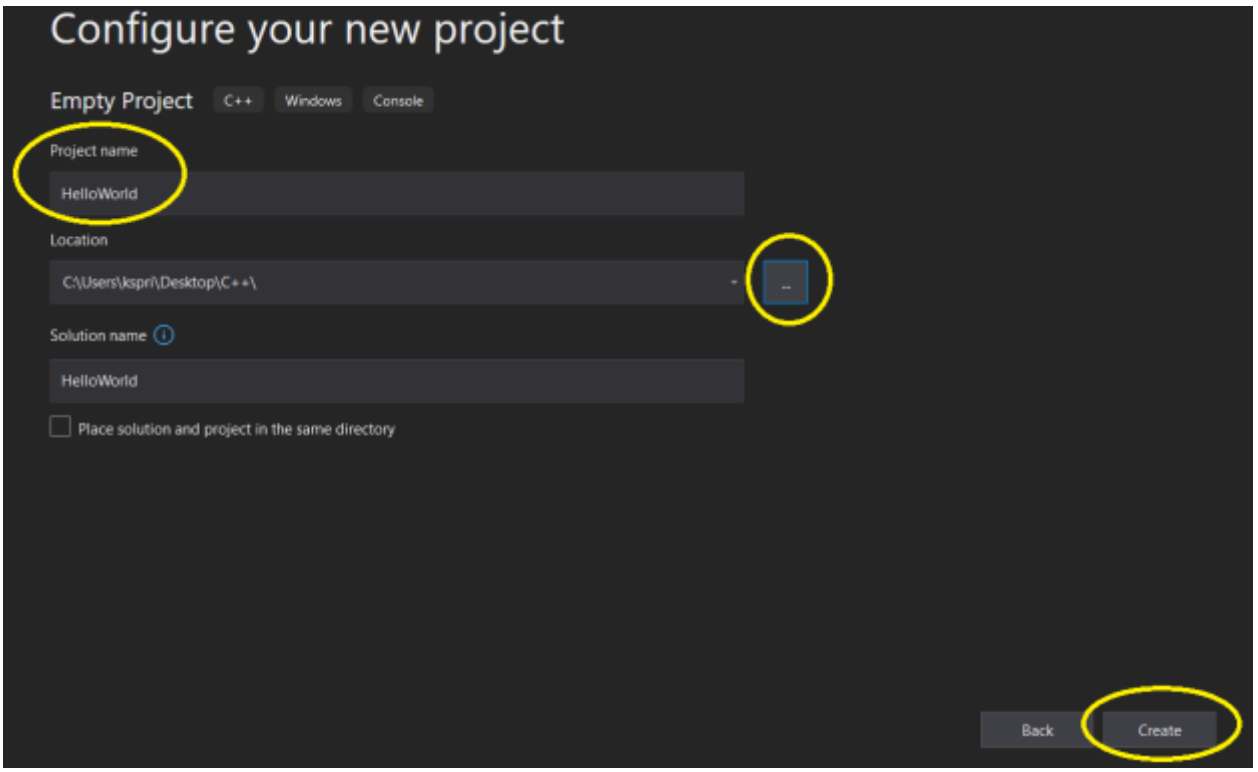

Visual Studio 2019
Open recent
As you use Visual Studio, any projects, folders, or files that you open will show up here for quick access.
You can pin anything that you open frequently so that it's always at the top of the list.
Get started
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

After clicking on the Create New Project button let's choose the C++ language, choose 'Empty Project', then click 'Next'

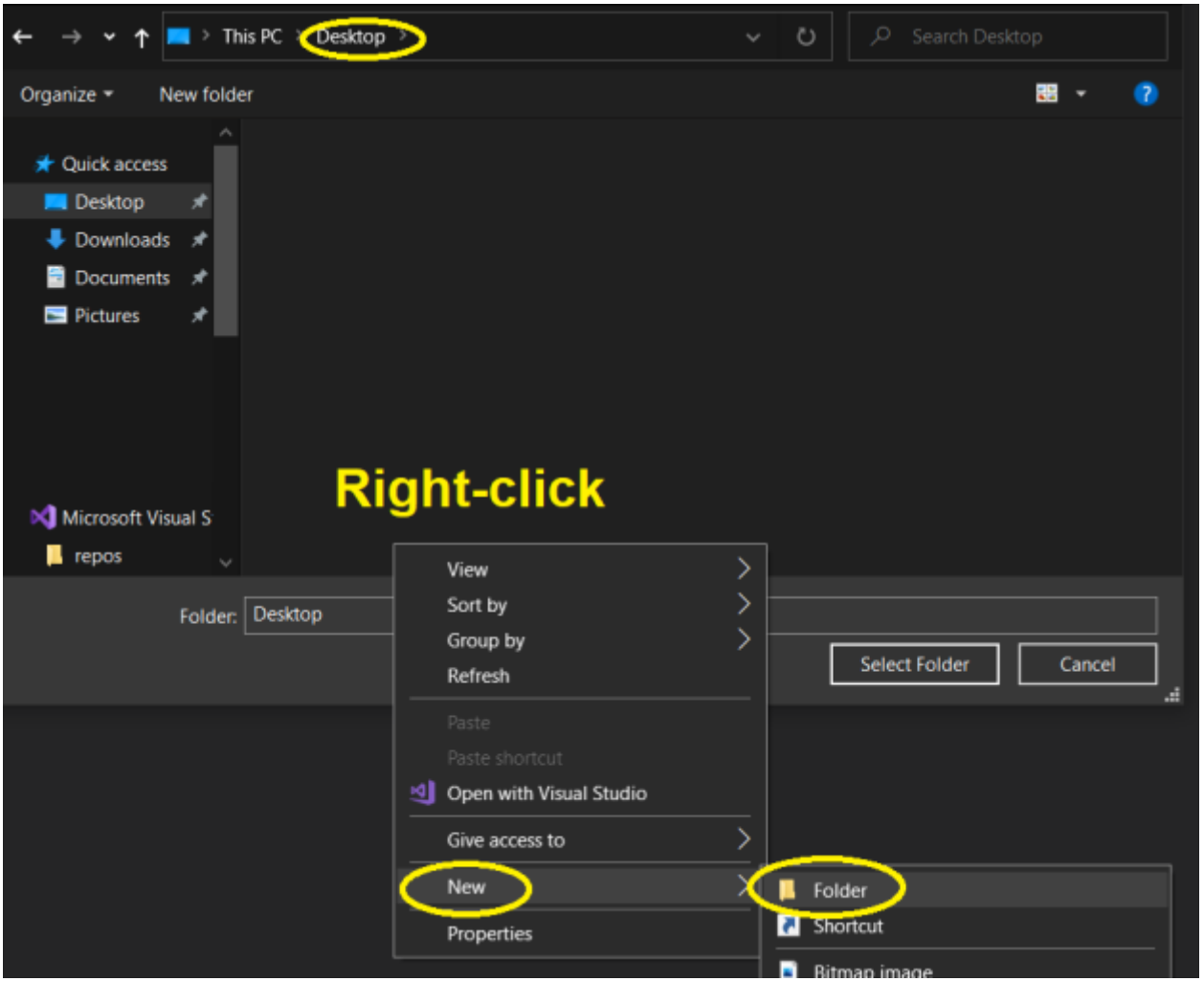




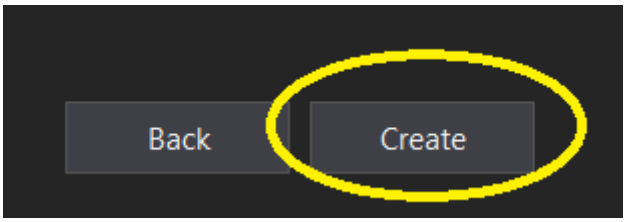
Now we must name our project and tell our IDE where to store the files. Let's name our project HelloWorld:



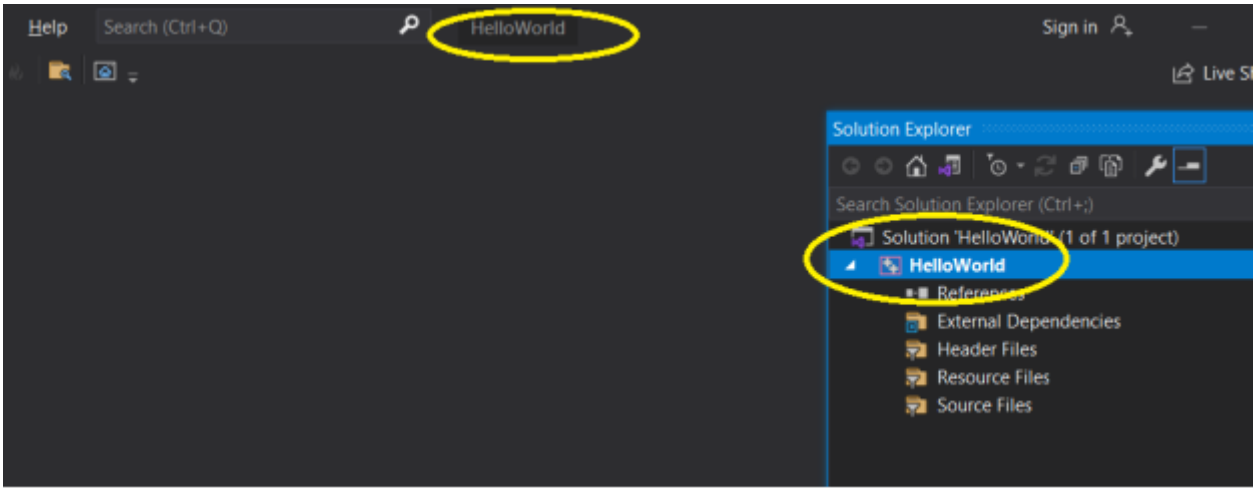
Next, we will need to click the three dots(...) next to the folder location option. Let's save our HelloWorld and all other projects we create in a new folder on our Desktop called C++. To create a new file on your desktop just right click within the Desktop folder, choose New and select Folder. Now name your new folder C++ folder and finally click save.



We have configured our new project so select the 'create' button:



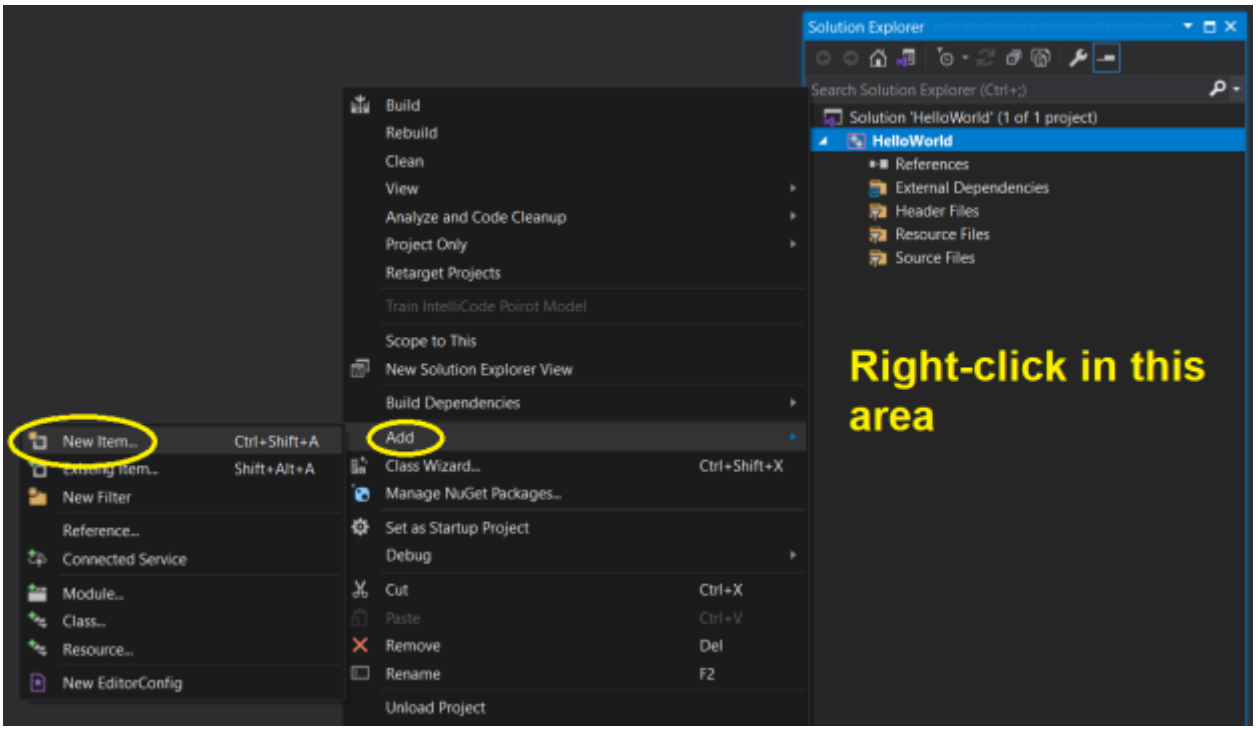
This will bring you to our main interface of our IDE with our HelloWorld project:


Help
Search (Ctrl+Q)
HelloWorld
Sign in
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'HelloWorld' (1 of 1 project)
HelloWorld
External Dependencies
Header Files
Resource Files
Source Files

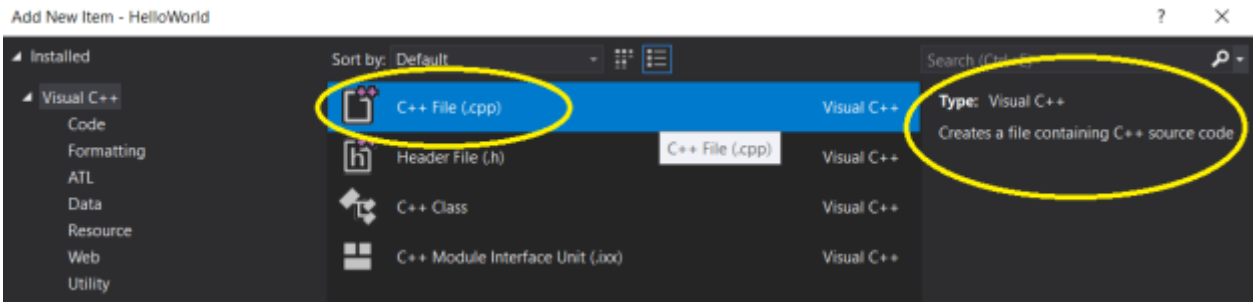
BUILD OUR FIRST CONSOLE PROGRAM

Console programs are programs that use text to communicate with the user and the environment, such as printing text to the screen or reading input from a keyboard. Console programs are easy to interact with, and generally have a predictable behavior that is identical across all platforms.

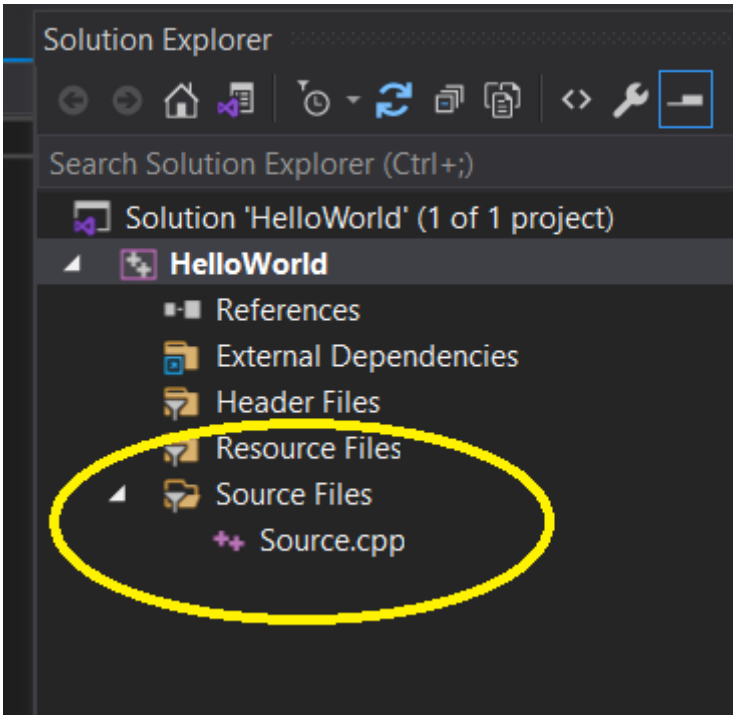
So now you are in your IDE interface.... Where do we put our code?! We need a c++ file so we can run our source code of course! Within our Solution Explorer let's right click, choose 'add', then select 'new item':



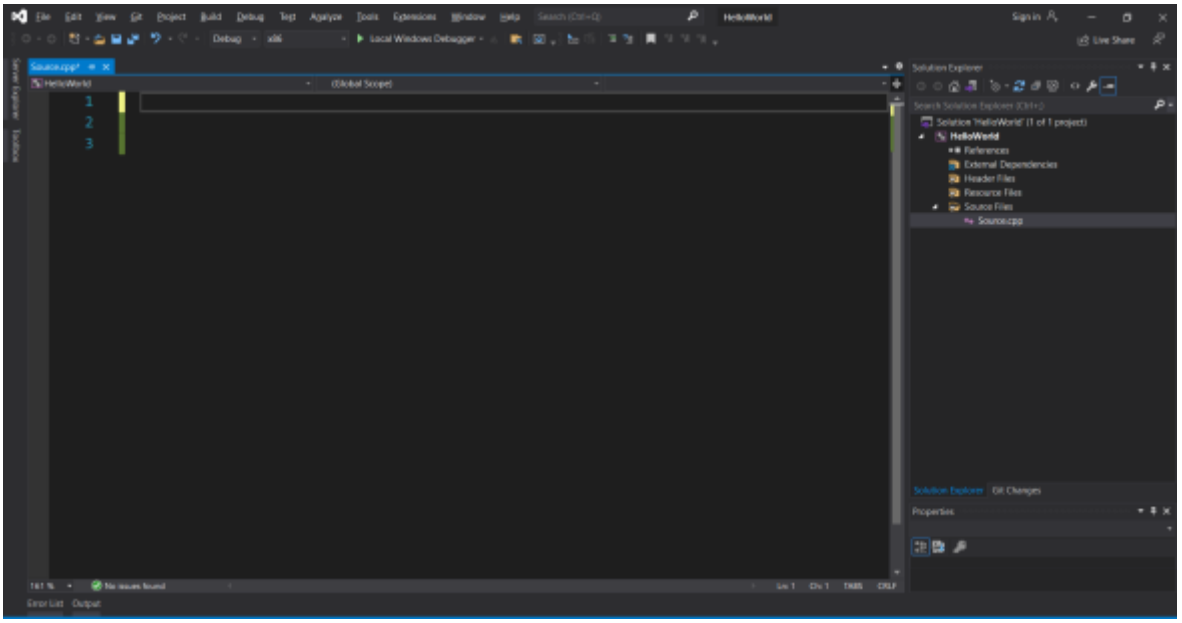
Then choose the C++ file:



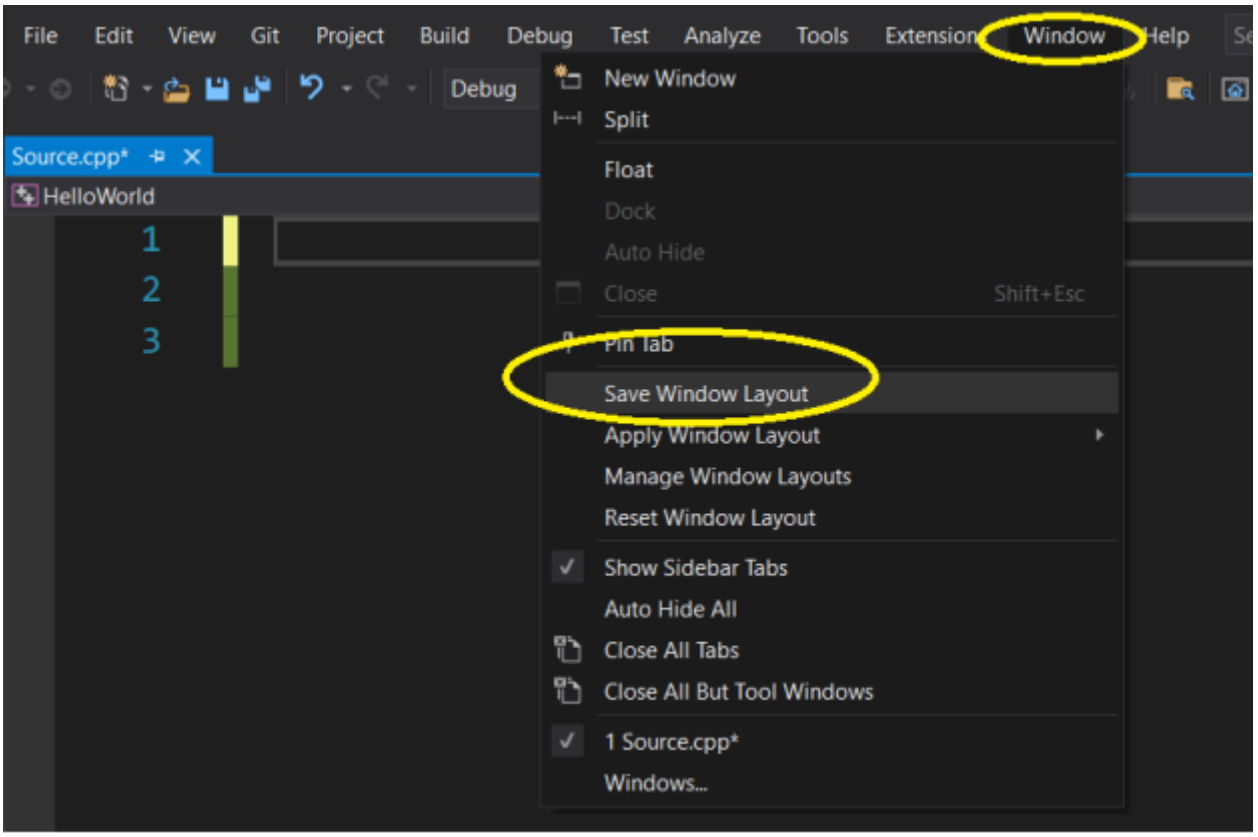
Now we have an actual file in our source files where we can start coding our program! As you can see, our C++ source files end in .cpp.



If you would like, you can drag and snap your different windows for your own personalized layout, then save that layout. I personally like this layout



To save this layout, click window and save layout, or in the future you can select apply layout:



STRUCTURE OF OUR PROGRAM

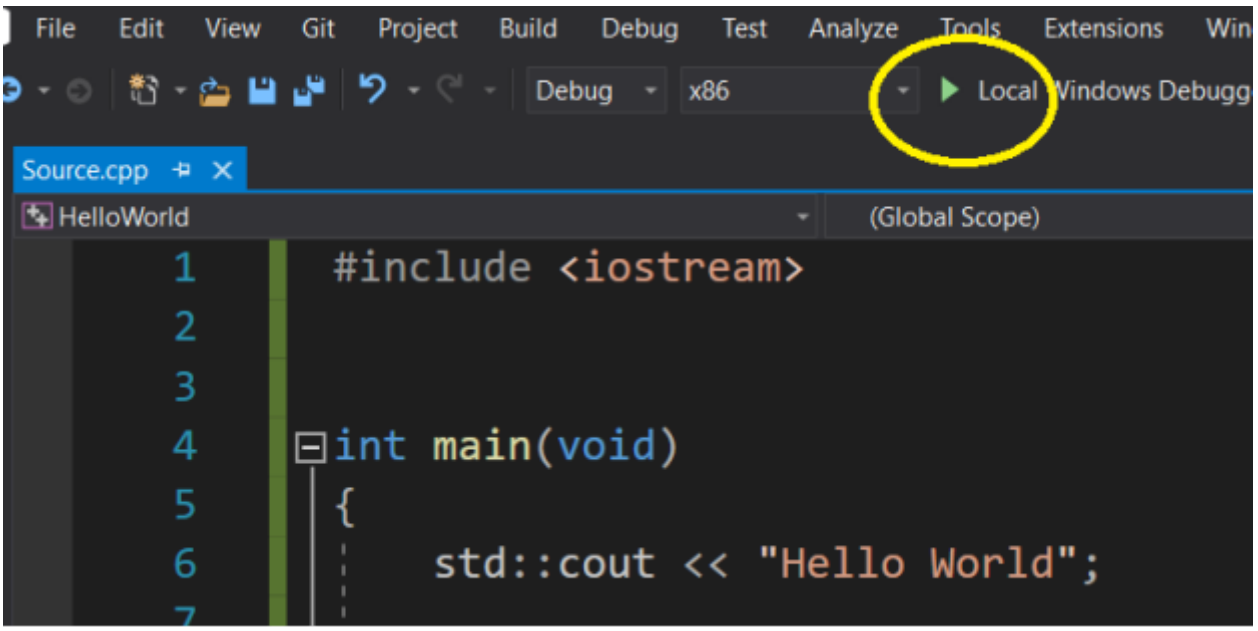
Copy/paste this code into your IDE:

_____

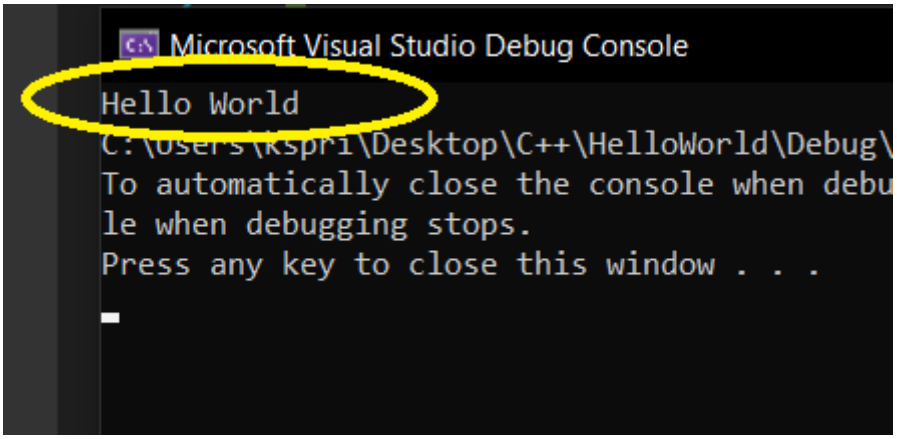
_____

```
int main(void)
    {
    std::cout << "Hello World";
    }
```

Now click the Run/Play button to run your program:



You should get a popup debugger console window that looks like this:


Microsoft Visual Studio Debug Console
Hello World
C:\Users\kspri\Desktop\C++\HelloWorld\Debug\
To automatically close the console when debu
le when debugging stops.
Press any key to close this window . . .

Your program did what it was supposed to and printed out Hello World! You have officially created your first C++ program and it ran successfully! On a side-note, if you would like to enlarge or decrease your font size just hold down the CTRL key and use your mouse wheel to change the font size. It helps to have larger font, especially when just learning! Now let's look at the code and actually understand what happened.

IOSTREAM

When a computer runs our code it will **always** read from top to bottom, in order. At the top of our code is where we will place any imports. What are these imports you ask? Well, the language of C++ has 'pre-built' codes that we can import and use.

Because the line starts with a #, it is called a preprocessor directive . The preprocessor actually reads your program before it is compiled. The #include causes the preprocessor to include the contents of the iostream file in the program, or whatever filename you place inside the <> brackets. The word inside the brackets is the actual name of the file that is to be included with your program.

Specifically, this iostream file contains a built-in code that allows our program to display output on the screen and to read input from the keyboard.

Remember when I said C++ has pre-built codes? Instead of you programming tons of lines to display output or read input from a keyboard, someone has already made that code! Now we can just include(import) that code, whew! Please note that iostream is just one pre-built code. There are tens of thousands of pre-built codes that we can use! These codes are in the C++ library - we will use some of them and we can even create our own. In summary, our header files are pre-built code files that we #include in whatever program we are creating.

THE MAIN

Our second line is the main! Everything happens in here and this must be in every single program. All code that we program happens or starts from within our main. The statement is also always the same, so you might as well memorize it now:

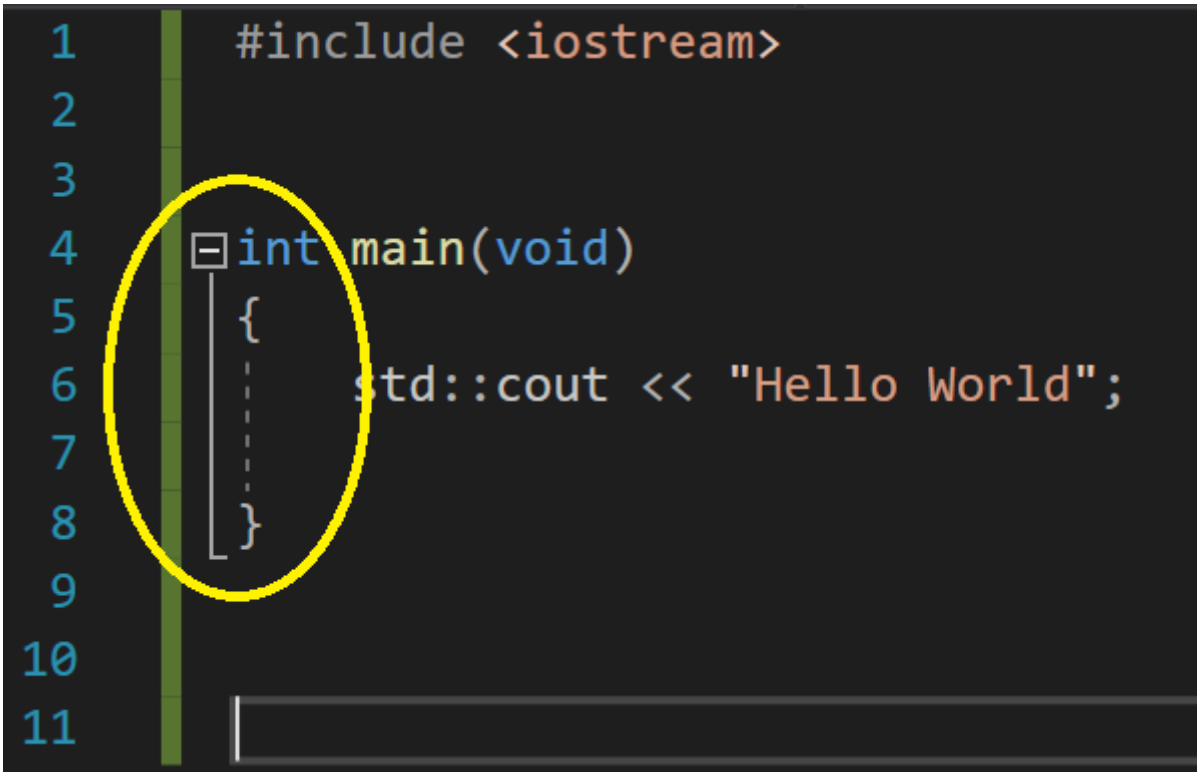
int main(void)

{

}

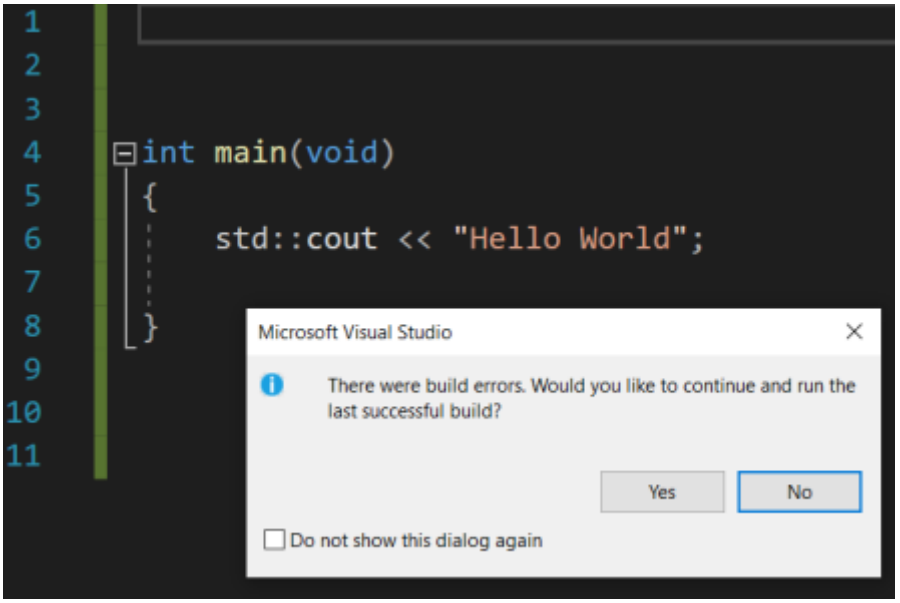
```
1  #include <iostream>
2
3
4  int main(void)
5  {
6      std::cout << "Hello World";
7
8  }
```

Anything within the braces { } of our main function will run in order, from top to bottom. Once the program reaches the closing brace } the program ends. Also, the opening and closing braces of our main method must be in line. If not, your program will not compile and run correctly. Our IDE helps tell us that our main method's braces are lined up correctly:

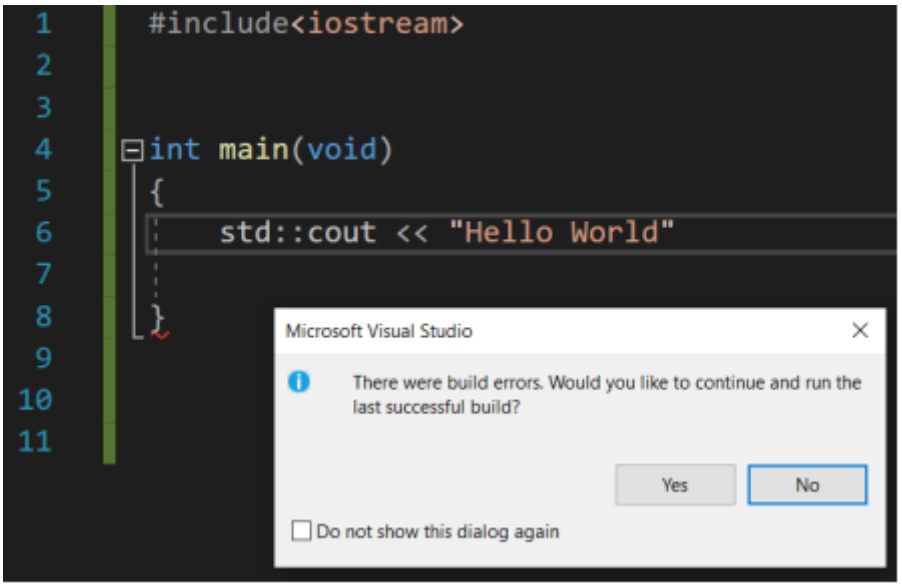


This is important because within our main function we will be creating our own functions with their own opening and closing braces { }. Now, within our main function we have our statement that prints out Hello World. The iostream library contains a few predefined variables for us to use. One of the most useful is std::cout, which allows us to send data to the console to be printed as text. cout stands for “character output”. Now you can see why we needed to #include that iostream file! If we had not included the iostream, our program would never know what to do with the std::cout statement.

Let’s delete the #include statements and see what happens when we run the program:



We are also using the insertion operator (<<) to send the text Hello World to the console to be printed. At the end of our statement there is a semicolon (;). This states that the end of a statement has happened and must always be there. If we were to remove the semicolon our program would not run:



DATA TYPES

Now that we understand our HelloWorld program, let's learn about the different a couple of data types that we can print out and display. The main data type is a string. A string is a collection of characters, or what we know as text and they always have quotation marks surrounding the string. Strings are a type- just like it sounds, a type is a way for the computer to understand what kind of input we are giving it. There are other types, like:

integers (whole numbers)

booleans (true or false)

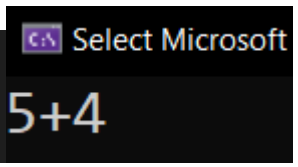
string(text)

char(a single character, a letter)

double(number with a decimal point/not a whole number)

And more, but don't worry about them yet! We will learn about them later. Lets try printing out a string that says 5 + 4:

```
std::cout << "5+4";
```



Did you get the same output? The program did what it should and printed out a string! Let's remove the quotation marks from our print statement and see what happens:

```
std::cout << 5+4;
```

The correct output is 9, which is an integer (whole number). As soon as we remove the quotations, the compiler is going to recognize that there is no string text and now assumes that the 5 and 4 are integers and needs to do a calculation because of the + operator. This shows you that data types are super important and can change the entire way your program runs if you do not understand them. Now that you know a couple data types, let's assign them to variables and use them!

VARIABLES

It is a perfect time to talk about variables. Variables are another important part of coding because we use them constantly! It will also be great to get comfortable with them before we move onto the next section. A variable is just a fancy name for a label tag or a way to keep track of information. It's just like many tags we see in life:

If someone were wearing a name tag you would know what their name is. Their name would be their Variable. Their name is actually stored in that name tag variable.

Then you can have multiple variables. Think of a nutrition label. The variables on a nutrition label would be cholesterol, grams of sugar, protein, carbs, calories, vitamins, etc.

It is interesting to see coding concepts like this in real life. There are examples of variables around us constantly. You probably did not know you would

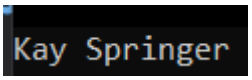
already be familiar with variables, huh?! Let's put that knowledge to good use in C++ programming!

When we code, we use variables to hold pieces of information for us - and just like clothing tags and food labels, coding variables can hold many kinds of information, such as strings, numbers, lists, and more.

So how do we create one? Let's make a variable to keep track of the name of this book's author (that's me!l). We would create a variable like this:

std::string authorOfThisBook = "Kay Springer"

We define that a variable is a string data type with the std::string word. That's all there is to it! We must first declare the data type of the variable, name the variable, then assign data to it! The variable authorOfThisBook now holds the text string Kay Springer. Now let's print out the variable authorOfThisBook:



Here's what is happening.

When we create the variable, we give it a descriptive name: authorOfThisBook. We could have named it just author as well. We name it with a descriptive name because it helps us remember what kind of information the variable is holding.

Next, we type an equal sign (=). This tells the computer that we are giving the author variable some information it should keep. This is called assignment or assigning a variable, in programming. Next, we type out the information our variable is supposed to hold. Finally, we must always end any statement with a semicolon(;). In this example, the variable author is holding Kay Springer.

In C++, whole text is known as strings and the data type you would declare a string variable is:

std::string

Now let's get your name involved! We will create another variable called reader. Go ahead and declare, name and assign your data to this variable. For this example, I will use the name John, but you should use your name.

std::string reader="John"

On the next line, use our std::cout print function to write your variable to the console. Your final code should look something like this:

reader = "John"

std::cout<

```
4   int main(void)
5   {
6       std::string authorOfThisBook = "Kay Springer";
7       std::string reader = "John";
8       std::cout << reader;
9
```

Now press ENTER. What happened? Do you see your name in the console window?

```
John
```

Now, here's the cooler part about variables: Let's say you share this book with your friend Alex. Obviously, our reader variable would now be incorrect because it should be your friend's name! So let's change the reader variable so it is assigned to your friend's name, but change nothing else. With the change, your code should look like this:

reader = "Shawn"

std::cout<

Now when you press Enter your friend's name should print out. This happened because you told the computer to assign Shawn to the reader variable thus overwriting the previous string of Bob. Using variables like this can really save time.

What happens when you write this code?:

std::cout<

reader "Shawn"

Are you surprised that your program would not run? A program always runs and reads code from top to bottom, so in this case we did not assign Shawn to the reader variable until

after the program tried to print the reader variable. That reader variable never existed until afterwards.

The variables we just used held strings (text), but as mentioned earlier variables can also hold other types of data. If we wanted to create a variable to hold our favorite number, how would we do it? In C++, whole numbers are known as integers and the data type you would declare a variable is:

int

So type this code and run your program:

int myFavoriteNumber = 7

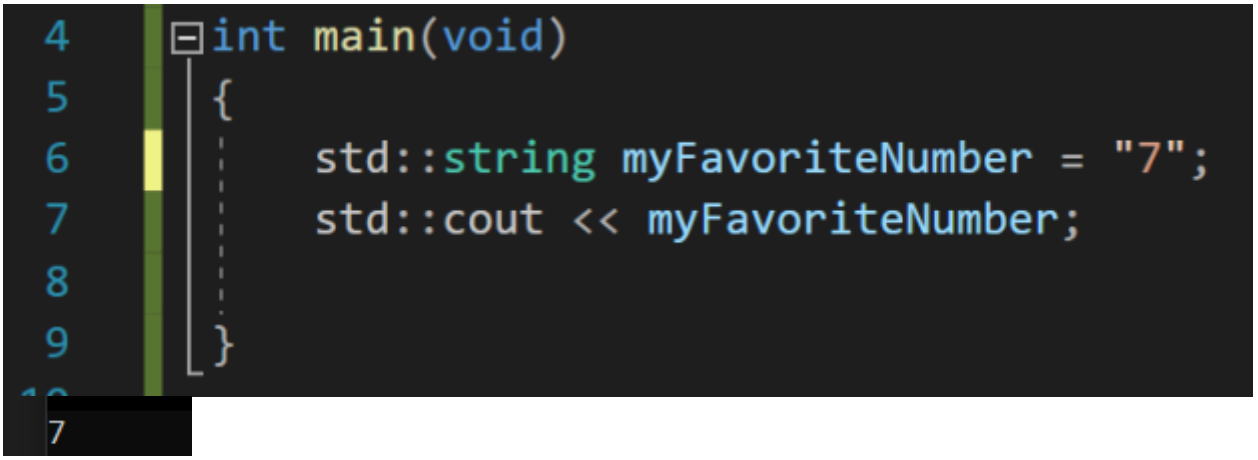
```
4   int main(void)
5   {
6       int myFavoriteNumber = 7;
7       std::cout << myFavoriteNumber;
8
9   }
```

```
7
```

We create this in the same way as before except without using quotation marks and we must change the data type of our variable. Only text strings have quotation marks around them. This is because the computer understands that when we place quotation marks around data then it is a text string type.

Whenever we deal with integers we simply type them out as a plain number. You never use quotes around them because it will confuse the computer into thinking you are using a string! The variable type must match the data we are trying to store in that variable. However, we can turn this 7 into a string type by placing

quotes around it. Modify your code by changing the variable type to declare a string, then place your quotes around the 7:



As you can see, we are able to use integers as numbers or string text! Variables will be used ALOT in our coding activities. Here are some good practices to keep in mind whenever you are creating variables.

Variables Cannot Start with a Number: When naming a variable, you want to be as descriptive as possible, but also follow the rules of C++. One of those rules is that variable names cannot start with a number. Try creating one, and see what happens:

int 100 days_of_health = 100

Did you get a syntax error? C++ doesn't like numbers in variable names and it gets really confused!

Variables Should Have the Same Styling: There are all kinds of ways to write your variables. The most important thing to remember is to pick one way and stick to it. You can write your variables using all lowercase letters, however, the accepted universal coding world usually uses camelCase. This is where the first word is always lowercase, then the first letter of any word after that is capitalized.

Example: numberOfHouses

TROUBLESHOOTING TIP: Why do we connect words or use underscores between words? This is because C++ doesn't recognize spaces in variable names. We either have to connect words (nospaceatall or NoSpaceAtAll) or use underscores to connect them (underscores_between_words). If you use spaces, you will get an error message.

Variables Should Have Meaning: Lastly, variable names should be as descriptive as possible. This means that when you read your code, you should know right away what your variable is and what kind of data it is storing. You should be able to understand it.

Here is an example list of good variable names:

```
mood = "happy"
age = 31
favoriteColor = "blue"
numberOfBooks = 2
```

And here's a list of not-so-good variable names:

```
A = 9
num pens = 13
CurDay = "Thursday"
FAvorItE_DrInk = "coffee"
```

See the difference? Clear names with meaning and style consistency are your best bets for great variable names.

CONCATENATION - STRING LITERALS

Strings are more useful when we can change or manipulate certain parts of them.

std::cout<<"I am so happy to learn C++ coding!"

Imagine that you have another emotion about learning to code in C++. How would you change this code to reflect your mood?

With concatenation of course!

Using concatenation, aka, string literals gives us an easy way to replace parts of a string or change their order. To do this, we create a variable, then we can place that variable in a string by literally using an addition sign (+) before and/or after whatever variable we want to insert in our string output. The addition operator(+) is what connects your variable to your output. Write the code below and run it:

int main(void)

{

std::string myFeeling = "excited";

std::cout << "I am so"+myFeeling+"to learn C++ Coding!";

}

```
4   int main(void)
5   {
6       std::string myFeeling = "excited";
7       std::cout << "I am so"+myFeeling+"to learn C++ Coding!";
8
9   }
```

Here is our

```
I am soexcitedto learn C++ Coding!
```

Uh-oh! The words are running into each other! Let's go back to our code and make sure to put a space at the end of the first string and at the beginning of the second string:

```
int main(void)
{
std::string myFeeling = "excited";
std::cout << "I am so "+myFeeling+" to learn C++ Coding!";
}
```

```
4   int main(void)
5   {
6       std::string myFeeling = "excited";
7       std::cout << "I am so "+myFeeling+" to learn C++ Coding!";
8
9   }
```

New output:

```
I am so excited to learn C++ Coding!
```

Much better! Now let's make 2 variables and place them into a sentence "I am so—feeling—to eat—food—!" Try it out, run your program and see what you come up with! I will show my output below:

```
4   int main(void)
5   {
6       std::string myFeeling = "upset";
7       std::string food = "oysters";
8       std::cout << "I am so "+myFeeling+" to eat "+food+"!";
9
0   }
```

```
I am so upset to eat oysters!
```

How did you do? Also, on a side note, if we were to take the exclamation mark off of our string statement we would only need the (+) before the food variable, not after.

PRINTING STRINGS & INTEGERS

Let's take the I can't wait to eat—number——favorite food plural —

Create your string variable and your integer variable, code your print statement, then run your program.

Did you run into a problem? C++ does not like it when you throw an integer variable into your output string!

```
4   int main(void)
5   {
6
7       std::string foodPlural = "donuts";
8       int someNumber = 209;
9       std::cout << "I can't wait to eat " +someNumber+ foodPlural;
10
11
12  }
```

```
donuts
```

Here is what we have to do... to isolate and print our integer variable in a string we must type << before the variable. This will print out your sentence fine, but now we need a " " space between the integer and string variable. Lastly, we need the + operator between the actual string and the following string variable for everything to print out correctly:

```
4   int main(void)
5   {
6
7       std::string foodPlural = "donuts";
8       int someNumber = 209;
9       std::cout << "I can't wait to eat " << someNumber << " " +foodPlural;
```

```
I can't wait to eat 209 donuts
```

MULTI-LINE STRINGS

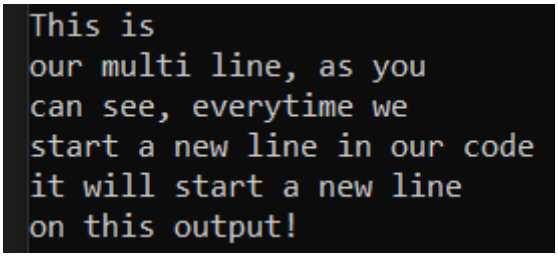
Maybe we don't want all of our string text on one line, maybe we want each sentence on a different line. We can write a multi-line sentence the exact same way as a normal string except you use a capital R (stands for Raw String Literal) followed by four quotation marks (") and followed by an opening parenthesis. End your multi-line string with a closing parenthesis, four quotation marks and a semicolon.

std:string myVariable = R""""(here's a sentence
then I press enter to go to next line
and enter to go to the next line)"""";

Here is what using the raw string literal multi-line looks like:

```
4   int main(void)
5   {
6       std::string myMultiLine = R"""(This is
7   our multi line, as you
8   can see, everytime we
9   start a new line in our code
10  it will start a new line
11  on this output!)""";
12
13      std::cout << myMultiLine;
14
15  }
```

Output:



Now let's try and add a string variable and an integer variable to our output.

PRACTICE ACTIVITIES

Data Types)

Activity #1 - Introduce Yourself

Code and print out a line that uses your name as a variable and says:

Hi! My name is Scott, I am very happy and I love to program

Do not look at the next page until you have completed the activity!

```
4   int main(void)
5   {
6
7       std::string myName = "Scott";
8
9       std::cout << "Hi, my name is " + myName + ", I am very happy and I love to program";
10
11  }
```

```
Hi, my name is Scott, I am very happy and I love to program
```

Activity #2 - Change Your Mood

Using your code from Activity #1 change your mood from happy to a different mood

---

Do not look at the next page until you have completed the activity!

```
4   int main(void)
5   {
6
7       std::string myName = "Scott";
8       std::string myFeeling = "excited";
9       std::cout << "Hi, my name is " + myName + ", I am very "+myFeeling+ " and I love to program";
10
11  }
```

```
Hi, my name is Scott, I am very excited and I love to program
```

Activity #3 - Change An Additional Mood

Using your code from Activity #2 add an emotion variable, then use that variable to replace the word “love” in your printed statement.

Do not look at the next page until you have completed the activity!

```
4   int main(void)
5   {
6
7       std::string myName = "Scott";
8       std::string myFeeling = "excited";
9       std::string myFeeling2 = "am happy";
10      std::cout << "Hi, my name is " + myName + ", I am very "+myFeeling+" and I "+myFeeling2+" to program";
11
12  }
```

```
Hi, my name is Scott, I am very excited and I am happy to program
```

Activity #4 - Create A Mad Lib Sentence

Create variables for your favorite food and a number
Create a sentence that says: I really like —favorite food—and I try and eat it at least—number—times a day!

Do not look at the next page until you have completed the activity!

```
4   int main(void)
5   {
6
7       std::string favoriteFood = "pizza";
8       std::string someNumber = "489";
9
10      std::cout << "I really like " + favoriteFood + " and I try and eat it at least " +someNumber+" times a day!";
11
12  }
```

```
I really like pizza and I try and eat it at least 489 times a day!
```

ARITHMETIC OPERATORS

You already learned about integers when we played with variables, now let's see different ways we can use integers with operators.

Arithmetic Also known as the math operators, arithmetic operators are used to perform the basic functions of math. As you will see in the following

chart most arithmetic operators work like they do in regular mathematics with a few exceptions

_____

_____ _____

_____

_____ _____

_____

_____ _____

_____ _____

_____ _____

_____ _____

Order of A special set of rules are followed for arithmetic operators This set of rules is called the order of operations. It is the proper order in

which arithmetic operations should be calculated. Let us pretend that you have many calculations for a single variable. For example, how we calculate our total when eating out at a restaurant:

total=20 + (20 * .0825) - 1.5 + 3

Here, there is a sales tax (the multiplication of .0825) added to our meal price (first addition calculation, $20), a coupon (the subtraction of $1.50),

and a tip for the waiter (the second addition calculation, $3). We can find the proper total by using the order of operations.

1. **Parentheses**

In calculations like this, the computer always calculates any expression in parentheses first. Parentheses. In our example, we would calculate the sales tax first (the.0825 represents sales tax of 8.25 percent). In this calculation, 20*0825 would equal 1.65

2. **Exponentiation**

The next calculation that is performed is exponentiation When the computer sees the ** operator, it raises one number to the power of another. What this means is, if you type 2 ** 4 in your shell, you'll get 16, because 2 to the power of 4 (also 2 x 2x2x2) equals 16. Because Our dinner calculation does not have any exponentiation calculations, we move onto the next operation of importance.

3. **Multiplication and Division**

In the order of operations, multiplication and division are next. They have the same level of importance as each other, so if both a multiplication and division calculation appear in the same line, we start with the calculation on the left and work our way to the right. For example, in this calculation:

4 * 3/ 2

We would first calculate 4 * 3 (which would be 12), since it's the calculation at the very left. Then, we would calculate the resulting 12 / 2, since we have worked our way to the right the final answer would be 6. Since our dinner calculation does not have any other multiplication or division calculations (except for

the one we already calculated in the parentheses step), we can move to the next rule of importance.

4. **Addition and Subtraction**

The calculations with the least importance are addition and subtraction This means they are performed last. So far, our total dinner bill

now looks like this, with the sales tax calculated first because it was in parentheses.

int total = 20 + 1.65 1.5 +3;

Now, we're left with a few addition and subtraction calculations. Since addition and subtraction have the same level of importance, we use the left-to-right order of calculating them, just like we did with multiplication and division. Let's see the remaining steps

First add 20 to 1.65, which would equal 21.65

Next, subtract the 1.5 from 21.65, which would equal 20.15

Finally, add the last calculation of 3 to 20.15 for a total of 23.15

Keep in mind, all the steps above aren't actually going to show in your shell. We simply went through the same steps that the computer takes to see how it actually calculates things.

Let's try printing this out in our IDE:

```
int total = 20 + (20 * .0825) - 1.5 + 3;
std::cout << total;
```

```
4   int main(void)
5   {
6
7       int total = 20 + (20 * .0825) - 1.5 + 3;
8
9       std::cout << total;
10
11  }
12
```

What was your output?

```
23
```

Wait, wasn't our output supposed to be 23.15?! We declared our total variable to be of int type, which is a whole number! Ok, let's change it to the float type because float types can be decimals!

```
4   int main(void)
5   {
6
7       float total = 20 + (20 * .0825) - 1.5 + 3;
8
9       std::cout << total;
10
11  }
```

```
23.15
```

There we go, much better and more accurate when we need a very accurate number, not just a whole integer number!

COMPARISON OPERATORS

Comparison The next set of operators we use in programming are called comparison operators. Just like their name, comparison operators help us

compare one value to another. When we use comparison operators, they return a True(1) or False(0) answer known as a Boolean type. Comparison operators and Booleans are very important because they help us make decisions in our code.

Let's go over the most used Comparison Operators:

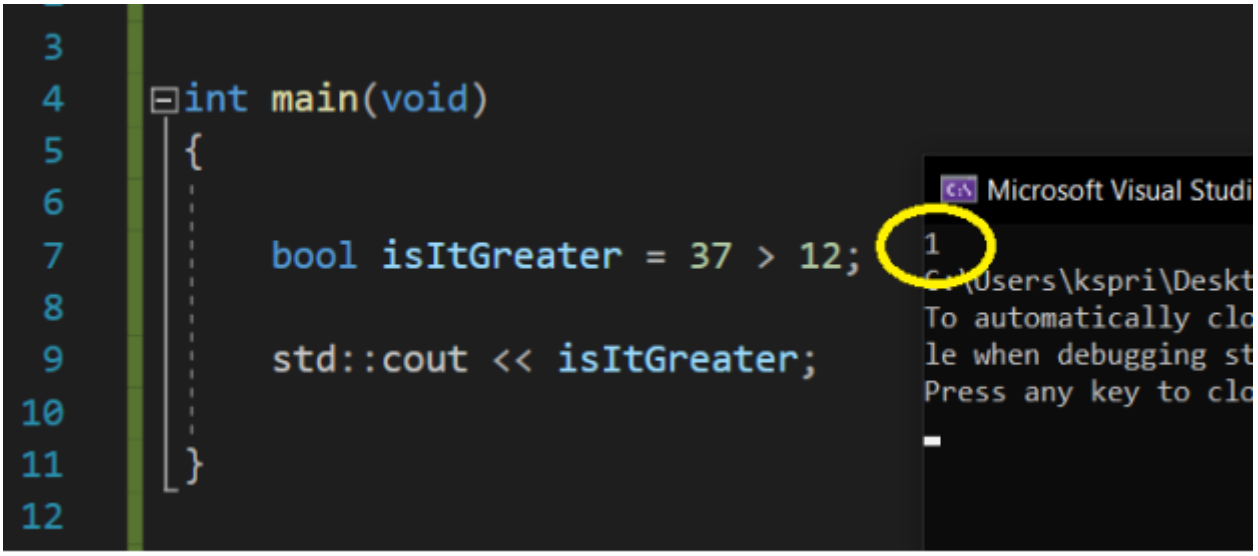
Greater Than

>

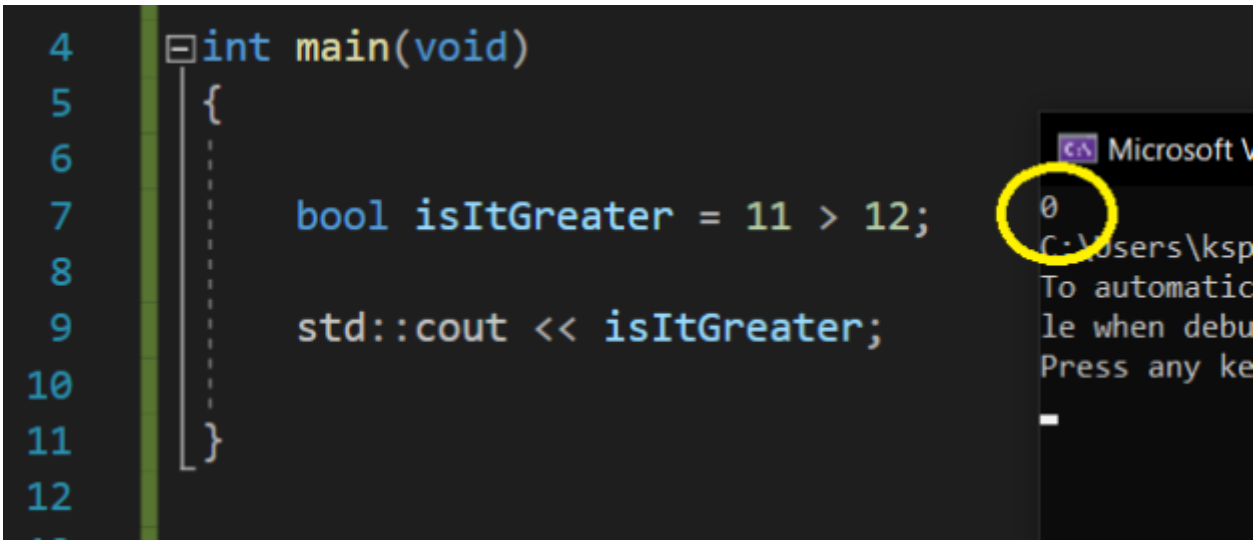
When you use it, the computer decides whether the value on the left side of the > symbol is larger than the value on the right side of the symbol. The isItGreater variable will hold a True or False value, which is why we declare it as a bool (boolean)

bool isItGreater = 37 > 12;

This variable is going to hold the answer to this question: Is 37 greater than 12? This returns a value of True(1) because 37 is greater than 12. Let's run this code:



Our output is 1 which represents True in boolean terms. Let's test that if we make this statement false we will get a False(0) output:



Since it is false that 11 is greater than 12 we get the expected output of False(0).

Less Than

<

When you use it, the computer decides whether the value on the left side of the < symbol is less than the value on the right side of the symbol.

bool isItLess = 37 < 12;

This variable is going to hold the answer to this question: Is 37 less than 12? This returns a value of False because 37 is in Not less than 12. Now if you were to print the variable isItLess it would output the value for False(0)

Greater Than Or Equal To

>=

We are trying to decide if the value on the left of the >= operator is greater than the one on the right OR if the value on the left is the same as the value on the right. Only one of these cases needs to be true in order for the computer to decide that the entire expression is True(1). So, in code:

bool isItGreaterOrEqual = 4 >= 3;

What do you think this will return? True or False? Did you say True? You're right because 4 is greater than 3, we know that the greater-than operator is correct. So even though the second operator, the equal-to operator, is not correct (because 4 is obviously not the same as 3), the computer still returns True because at least one operator is correct (the greater-than action).

Next example;

bool isItGreaterOrEqual = 3 >= 3;

This one is also True! This time, the equal-to operator is correct, instead of the greater-than operator.

Here's one more:

isItGreaterOrEqual = 1 >= 3;

What do you think? False? That's right because BOTH operators are not correct. The number 1 is not greater than 3, so the greater-than operator is incorrect, and 1 is not the same as 3, which also makes the equal-to operator incorrect.

Less Than Or Equal To

<=

Hopefully you are starting to see a pattern here! This is the less-than-or-equal-to operator. Just like the greater-than-or-equal-to operator, we are making sure at least one of the operators is correct. For the less-than-or-equal to operator, we are looking at the values to see if the value on the left of the <= operator is either smaller than the one on the right of the operator or the same as the one on the right.

What do you think this code will return when you write it in your IDE?

bool isItLessOrEqual = 1 <= 3;

That's right! This returns True because 1 is smaller than 3. This makes the less-than operator correct, even if the equal-to operator is not. And since one of the operators is correct, the whole thing returns True(1).

How about this one?

bool isItLessOrEqual = 7 <= 7;

Same thing. This also returns True, as the equal-to operator is correct.

Equal To

==

This one is much more simple than the last two operators. Just like it sounds, it asks the computer to decide if the value on the left of the == symbol is the EXACT same as the value on the right of the == symbol. Easy!

How will this return?

23 == 22

False(0) because 23 is not equal to 22

What about this one?

10 == 10

True(1) because they are exactly the same

Here's A tricky one:

10 == "10"

False(0) because the 10 on the right is a string type and the 10 on the left is an integer/number type. Numbers will never equal or be the same as a text string.

Not Equal To

!=

Also like the name, the not-equal-to operator asks the computer to figure Out if the value on the left side of the!=symbol is not the same as the value on the right side of the != symbol. Go ahead and try to guess the next few examples.

5 != “five”

True(1) because the integer 5 is **not equal to** the string text “five”

10 != “10”

True(1) because the integer 10 is **not equal to** the string text “10”

4 != 3

True(1) because the integer 4 is **not equal to** the integer 3

9 != 9

False(0) because the integer 9 **is equal to** the integer 9

LOGICAL OPERATORS

Logical Logical operators are used to help us compare True or False operands. These are very helpful because they can make our decision making rules more complex, which means smarter code!

And Operator

The and operator checks that the values on the right and left of it are both True(1).

It looks like this: **&&**

If there's a point in our code that should only run when two conditions are met we should use the and operator. Let's say that we are looking for a specific pizza with pepperoni and mushrooms. Remember 0 is False and 1 is True:

bool pizzaHasPepperoni= 1;

bool pizzaHasMushrooms = 0;

To check that the pizza you were evaluating had both pepperoni and mushrooms, you would use the and operator like this

pizzaHasPepperoni && pizzaHasMushrooms

The and operator allows you to check both conditions: that the pizza slice has pepperoni and that it has mushrooms. Only then would you take that slice of pizza, only if both of your conditions were met! In this case there were no mushrooms... so your condition returned false and you do not get the slice with pepperoni and mushrooms.

Let's print out a simple statement that checks if our 2 variables are both true... we know they are not because we assigned values

to them, however, when we begin creating complex programs you are going to need to compare and get boolean answers for things that may constantly be changing.

```
4   int main(void)
5   {
6
7       bool pizzaHasPepperoni = 1;
8       bool pizzaHasMushrooms = 0;
9       bool areTheseTwoTheSame = pizzaHasPepperoni && pizzaHasMushrooms;
10      std::cout << areTheseTwoTheSame;
11
12
13  }
14
```

In our areTheseTwoTheSame variable we are asking if the 2 variables are both true. The answer to that will be stored as a boolean in that variable:

We get False(0) of course:

```
0
```

Or Operator

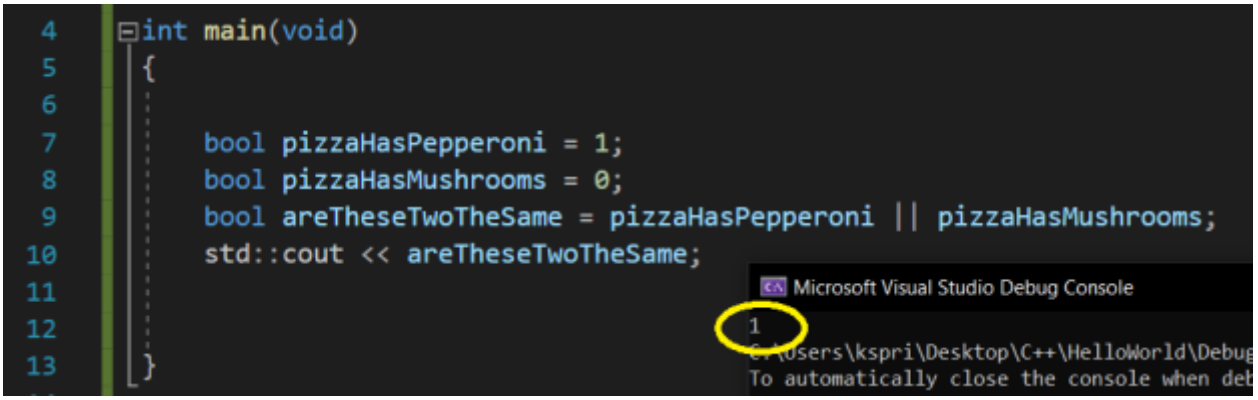
Ihe or operator checks to make sure that **at least** one value being compared is True.

It looks like this: **||**

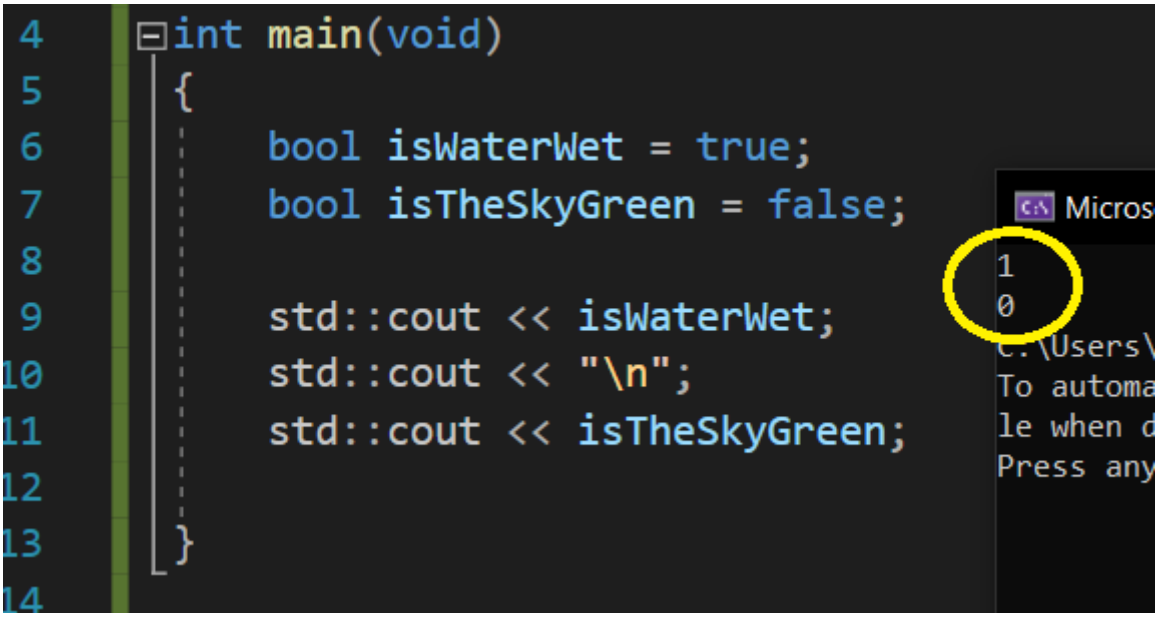
The or bar symbol is above your enter button. With our pizza example, we couldn't find any pizza slices that had both pepperoni and mushroom on it. But we still want pizza so you decide that if the pizza has either pepperoni **or** mushroom on it you will still select and eat that pizza. Here is where the or operator will come into play. To check for either pepperoni **or** mushroom, you would write code like this:

pizzaHasPepperoni || *pizzaHasMushrooms*

This way, if the pizza you were checking had either pepperoni or mushroom on it, you'd take it. Let's run the same code but instead place the or operator instead of the and operator. This should return a True(1) value because one of the values in question is true.



You can also set your booleans to true or false and the output will still be 1 for true or 0 for false. For example, let's set 2 booleans and see the output:



Notice that I printed out a "\n". This is called a new line and can be placed as its own, or I could put it at the end of a string output like so:

std::cout << "Jack and Jill \n";

std::cout << "went up the hill";

This would produce an output of Jack and Jill on one line and went up the hill on a second line:

In this section we have learned

There are two main numeric types we will work with most often: integers and floats

Operators are special keywords or characters that enable us to perform actions.

The first set we learned about were arithmetic operators. Arithmetic operators are similar to the ones we use in math and it is important to remember the order of operations to calculate something correctly.

The order of operations, from most important to least important,is: parentheses exponentiation, multiplication, division, addition, and subtraction (PEMDAS).

If there is more than one calculation with the same level of importance to evaluate, we go from left to right.

Another set of operators we learned about are comparison operators. These help us compare two values to each other. We have operators to compare if one value is greater than (>), less than (<), greater than or equal to (>=), less than or equal to (<=), equal to (==), and not equal to (!=) another.

Finally, we learned about logical operators, which help us make smarter comparisons

The and operator helps us determine if two expressions are both True.

The or operator checks that at least one expression we pass to it is True.

PRACTICE ACTIVITIES

(Operators & Booleans)

Activity #1 - Introduce Yourself & Your Age

Code and print out a line that uses your name and age that says:

My name is—name—and I am—age—years old

To calculate your age variable use this current year minus the year you were born, or whatever calculation makes you happy!

Do not look at the next page until you have completed the activity!

```
4   int main(void)
5   {
6
7       std::string name = "Jason";
8       int age = 2025-1924;
9
10      std::cout << "Hi, my name is " + name + " and I am " <<  age << " years old";
11
12  }
```

```
Hi, my name is Jason and I am 101 years old
```

Activity #2 - Find Out Who Has More or Less Mushrooms

Compare the amount of mushrooms on 2 different slices of pizza

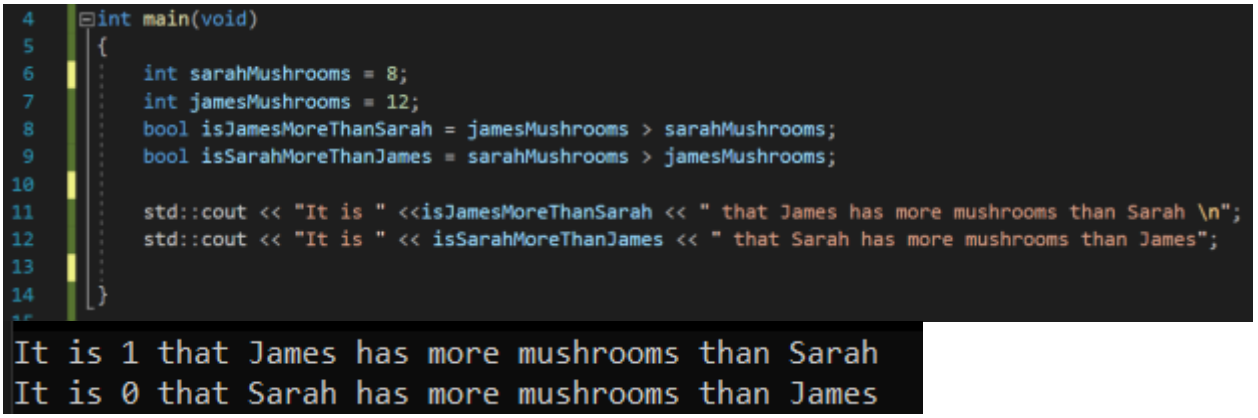
Create variables stating that James has 5 mushrooms on his slice and Sarah has 8 mushrooms on hers. Print out the following 2 outputs:

It is—true(1) or false(0)—that James has more mushrooms than Sarah.

It is—true(1) or false(0)—that Sarah has more mushrooms than James

*insert a new line between these 2 sentences - you do this by typing \n at the end of the first string, so after Sarah\n in this circumstance.

Do not look at the next page until you have completed the activity!



```
int main(void)
{
    int sarahMushrooms = 8;
    int jamesMushrooms = 12;
    bool isJamesMoreThanSarah = jamesMushrooms > sarahMushrooms;
    bool isSarahMoreThanJames = sarahMushrooms > jamesMushrooms;

    std::cout << "It is " <<isJamesMoreThanSarah << " that James has more mushrooms than Sarah \n";
    std::cout << "It is " << isSarahMoreThanJames << " that Sarah has more mushrooms than James";

}
```

```
It is 1 that James has more mushrooms than Sarah
It is 0 that Sarah has more mushrooms than James
```

NAMESPACE

So far we have been using std:: before declaring a string variable and every time we print something out. "std" (abbreviation for the standard) is a namespace whose members are cout, cin, endl, etc. Instead of doing all of this extra typing we can add using namespace std; to the header under our #include statement:

```
1   #include<iostream>
2   using namespace std;
3
4   int main(void)
5   {
```

Now let's make a couple strings and print them out:

```
1   #include<iostream>
2   using namespace std;
3
4   int main(void)
5   {
6       string name = "Paul";
7       string feeling = "hungry";
8
9       cout << name + " is very " + feeling + "!";
10
11  }
12
13
14
```

```
Microsoft Visual Studio Debug Console
Paul is very hungry!
C:\Users\kspri\Desktop\C++\HelloW
To automatically close the consol
le when debugging stops.
```

See how much easier it is to not type out std:: before string or cout?!

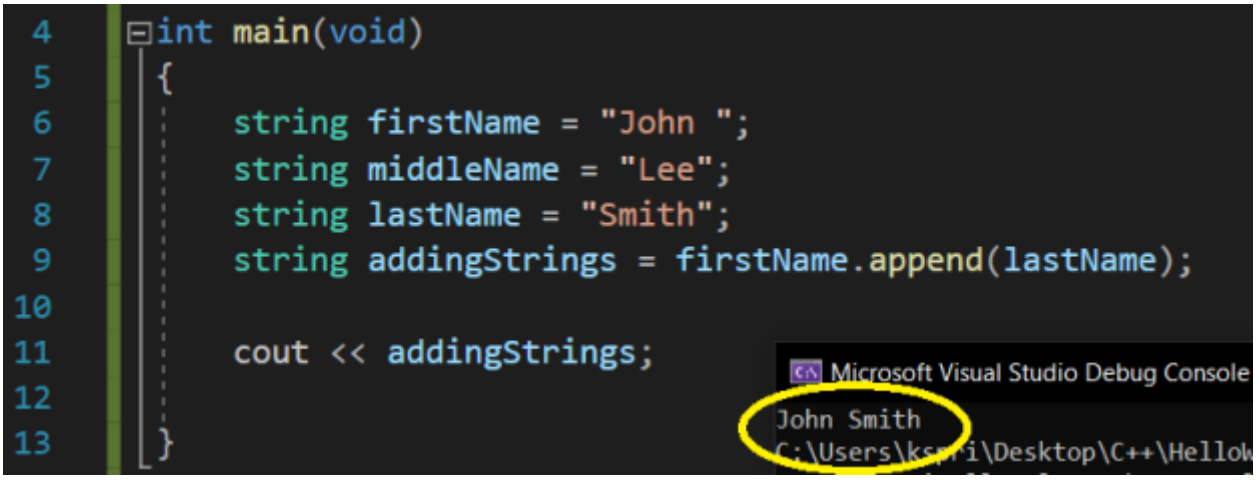
THE STRING OBJECT

We have already learned a lot about strings and it is important to know that a string is actually an object in C++! An object is something that has different built-in functions. We can use these functions to do different things to our strings!

Append() Function

Previously in concatenation, we were using the + sign to add strings together so you would get comfortable with how strings work. Now let's use an actual string object function called append(). The append() function adds or modifies an existing string.

Let's make 3 variables, one each for first name, middle name and last name. Then we will make a 4th variable where we will append the last name to the first name.



That is pretty straightforward, but what if we want to print all 3 variables in order? What would you do? Go ahead and try!

So obviously we want to append the middle name string to our first name string:

```
string addingStrings = firstName.append(middleName);
```

And since we can use the append() function on any string, let's add it to our middle name string, ***EXACTLY*** like we used it on our first name string!

```
string addingStrings = firstName.append(middleName.append(lastName));
```

Now, let's run our code:

length() and size() Functions

Another string object function is length() or size()! These functions get the length of a string and they both work the exact same way, so you can choose whichever is easiest to remember. The length of a string is literally how many characters are in a string. Let's check the length of our firstName string variable. Remember, to use a function on an object you must place a dot(.) followed by the function name and (). Let's make a string variable that holds all the characters of the alphabet then use the size() and length() methods:

```
4   int main(void)
5   {
6       string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
7
8       cout << alphabet.size();
9
10      }
```

Microsoft Visual Studio De

26

C:\Users\kspri\Desktop\

```
4   int main(void)
5   {
6       string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
7
8       cout << alphabet.length();
9
10      }
```

Microsoft Visual Studio De

26

C:\Users\kspri\Desktop\

Now what if we wanted to know exactly what is in the 3rd spot of our alphabet string? There is a way to do this with Access Strings.

Access Strings

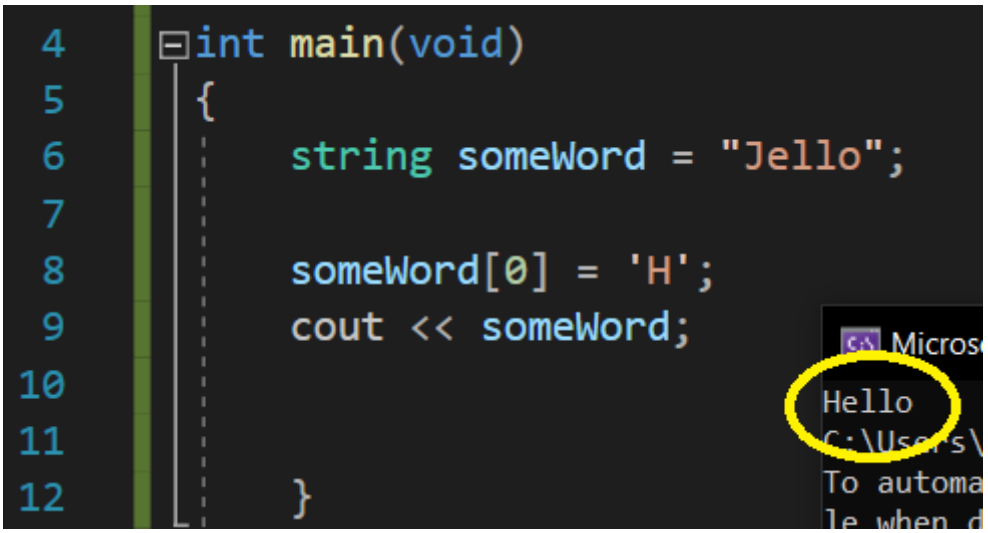
You can access characters in a string by the string's index number. In computer language an index always starts with 0 and then counts up from there. So for example, in our alphabet string variable you may think that the character 'A' would be at the first index. However, the first character of a string will always be at index 0.

Let's play with this and create code that will show what character is in these indices (plural for index). To do this we will write our variable name followed by brackets[ ]. You will place an index number within those brackets and it will return whatever is at that exact index number, easy peasy!

Now let's print out some more:

Changing String Characters

Now what if we wanted to change one or more characters in a string? I would like to create a string variable holding the value of Jello, then I want to change the word to Hello. First, we would create code that says, hey, go to the first index of Jello... then we would assign (=) a value of the character 'H' like so:

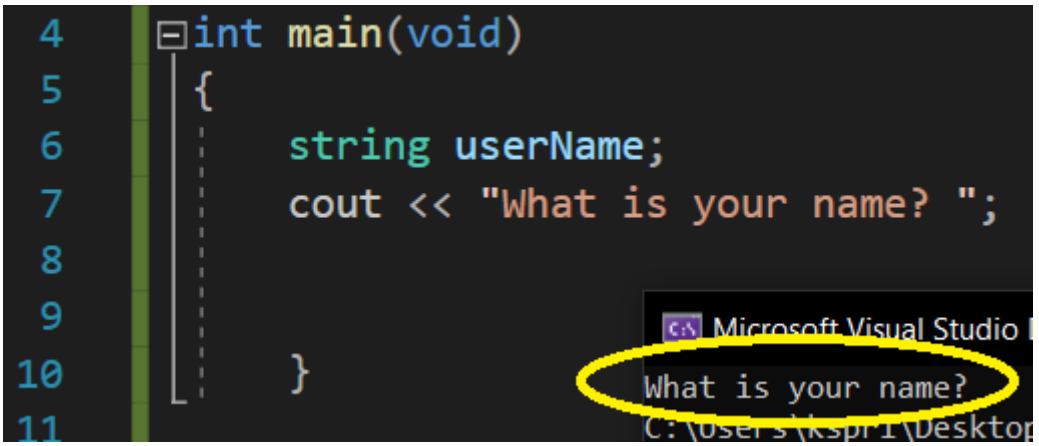


Did you notice that we placed apostrophe's (') around the 'H' instead of quotation marks? We do this because apostrophes indicate that this is a character, not a string. Just as quotation marks around a string indicate that it is a string, not a character or anything else.

USER INPUT

Guess what else we can do with our variables? Instead of assigning our own values to them, we can store our user's input! This is obviously a powerful situation because most programs heavily rely on and change based on the user's input. Think about a simple word document... in the background the computer program is taking each character you type, processing it, then displaying it as text on the document!

So how do we get input from a user? Well first we need to create a variable that holds no value, then ask the user for information that we can assign to that variable.

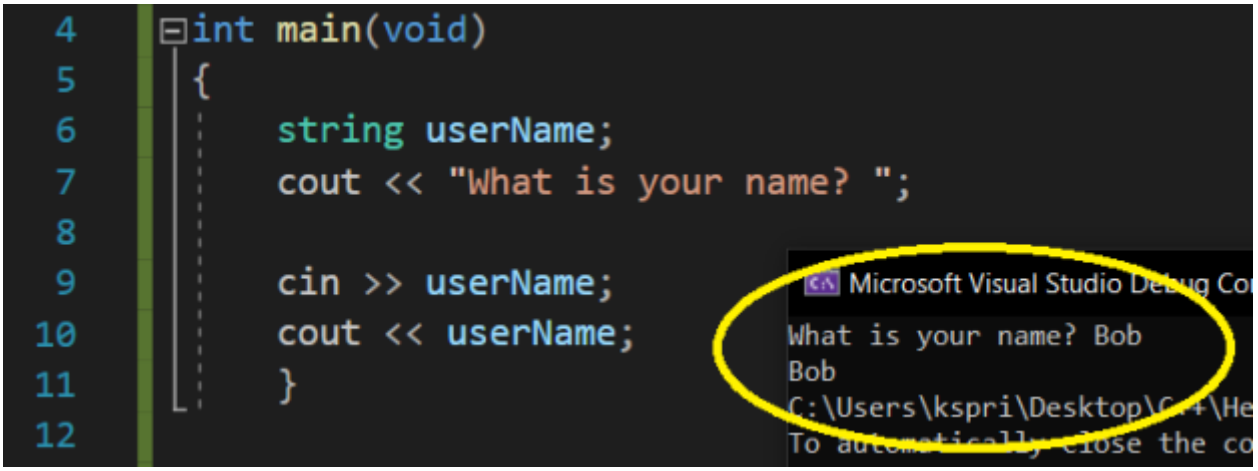


The user (which is you right now) will type their name. Now what?! How do we get what was typed into that userName variable? We use something called an extraction operator which grabs the input from the user's keyboard. It is similar to the print cout operator we have been using, but different. We use the word cin, followed by >>.

```
4   int main(void)
5   {
6       string userName;
7       cout << "What is your name? ";
8
9       cin >> userName;
10
11      }
```

Think of it like cin is grabbing the user input and the >> act like two little arrows pointing to a variable where we want that information stored. The extraction operator terminates recording user keyboard input anytime a space bar, tab, etc. is pushed.

Now run your program, type in your name, press enter and let's see if your program prints out your userName variable correctly.



Let's customize our message to the user. Make your message say ''Hi,—userName—, nice to meet you!

```
4   int main(void)
5   {
6       string userName;
7       cout << "What is your name? ";
8
9       cin >> userName;
10      cout << "Hi, " << userName + ", nice to meet you!";
11      }
12
13
14
15
```

```
Microsoft Visual Studio Debug Console
What is your name? Sarah
Hi, Sarah, nice to meet you!
C:\Users\kspri\Desktop\C++\HelloWorld\Debug\
To automatically close the console when debu
```

Did you get a similar output? Great! What if we type out our last name as well?

```
4   int main(void)
5   {
6       string userName;
7       cout << "What is your name? ";
8
9       cin >> userName;
10      cout << "Hi, " << userName + ", nice to meet you!";
11      }
12
13
14
15
```

```
Microsoft Visual Studio Debug Console
What is your name? Jane Smith
Hi, Jane, nice to meet you!
C:\Users\kspri\Desktop\C++\HelloWorld
To automatically close the console wh
```

In this example I typed Jane Smith but the cin extraction operator only grabbed Jane. Remember, the cin stops grabbing the user input if a spacebar or tab button is clicked. What we use to get around this is, you guessed it, another handy built-in function! Often when working with strings we use the getline() function.

getline() Function

The getline() function takes 2 parameters that you place in it's parenthesis (). It takes cin as the first parameter and the string variable as the second. To use this method we must first bring

in a package where the getline() function resides. To do this, go to the top of your code and add the following statement:

#include

```
1  #include<iostream>
2  #include <string>
3  using namespace std;
```

Now you can successfully use the getline() function:

getline(cin,userName);

Let's change that code we just ran and see what happens now:

```
5  int main(void)
6  {
7      string userName;
8      cout << "What is your name? ";
9
10     getline(cin, userName);
11     cout << "Hi, " << userName + ", nice to meet you!";
12     }
13
14
15
16
17
```

```
What is your name? John Smith
Hi, John Smith, nice to meet you!
To automatically close the console when debug
```

The getline() function is just grabbing what the user input as an answer to your cout<< statement and storing that information in a variable. If you never use the getline() function then the cout<< is just a message to the user and it ends there. On activity #2 below, you will dive into using this getline() function!

& User

Activity #1 - Short Mad Lib

Create variables for 2 adjectives, plural piece of clothing and a room in a house. Set your variables before copying the story below.

It was a—adjective—, December day. I
woke up to the—adjective—smell of—plural piece of clothing—roasting in the—room in a house—downstairs.

_____

Do not look at the next page until you have completed the activity!

int main(void)
{
string firstAdj = "lucky";
string secondAdj = "ridiculous";
string clothing = "socks";
string room = "laundry room";
cout << "It was a " + firstAdj + ", December day. I \n";
cout << "woke up to the " + secondAdj + " smell of " + clothing + "\n";
cout << "roasting in the " + room + " downstairs.";
}
Microsoft Visual Studio Debug Console
It was a lucky, December day. I
woke up to the ridiculous smell of socks
roasting in the laundry room downstairs.

Activity #2 - User Input For Short Mad Lib

Do the same thing as in the previous exercise but set all of your variables based on the user input, then display the full mad

lib to your user, along with a customized message using their name.

Remember, create variables for 2 adjectives, plural piece of clothing and a room in a house.

Hi—username—, here is your mad lib story!

It was a—adjective—, December day. I

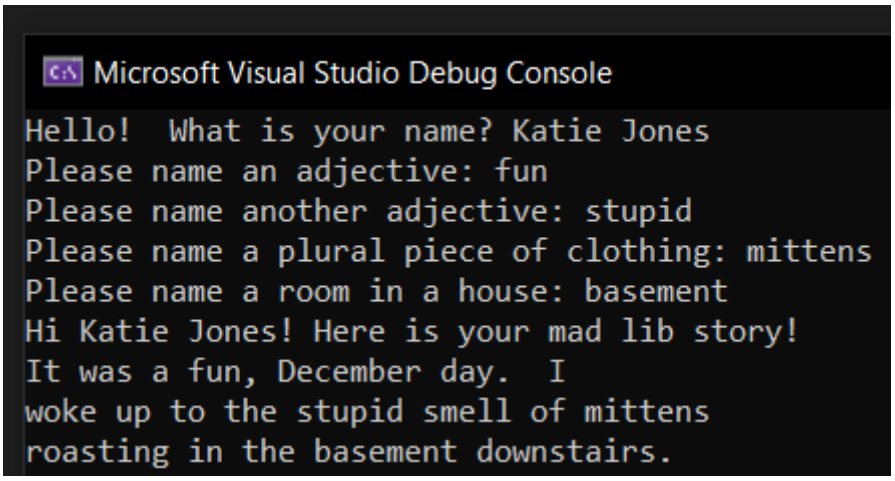
woke up to the—adjective—smell of—plural piece of clothing—

roasting in the—room in a house—downstairs.

*hint(remember, you want to get the line of what the user inputs and set it to a variable)

Do not look at the next page until you have completed the activity!

```
int main(void)
{
    string userName;
    string firstAdj;
    string secondAdj;
    string clothing ;
    string room ;

    cout << "Hello!  What is your name? ";
    getline(cin, userName);
    cout << "Please name an adjective: ";
    getline(cin, firstAdj);
    cout << "Please name another adjective: ";
    getline(cin, secondAdj);
    cout << "Please name a plural piece of clothing: ";
    getline(cin, clothing);
    cout << "Please name a room in a house: ";
    getline(cin, room);

    cout << "Hi "+userName+"! Here is your mad lib story!\n";
    cout << "It was a " + firstAdj + ", December day.  I \n";
    cout << "woke up to the " + secondAdj + " smell of " + clothing + "\n";
    cout << "roasting in the " + room + " downstairs.";

    }
```


Microsoft Visual Studio Debug Console
Hello! What is your name? Katie Jones
Please name an adjective: fun
Please name another adjective: stupid
Please name a plural piece of clothing: mittens
Please name a room in a house: basement
Hi Katie Jones! Here is your mad lib story!
It was a fun, December day. I
woke up to the stupid smell of mittens
roasting in the basement downstairs.

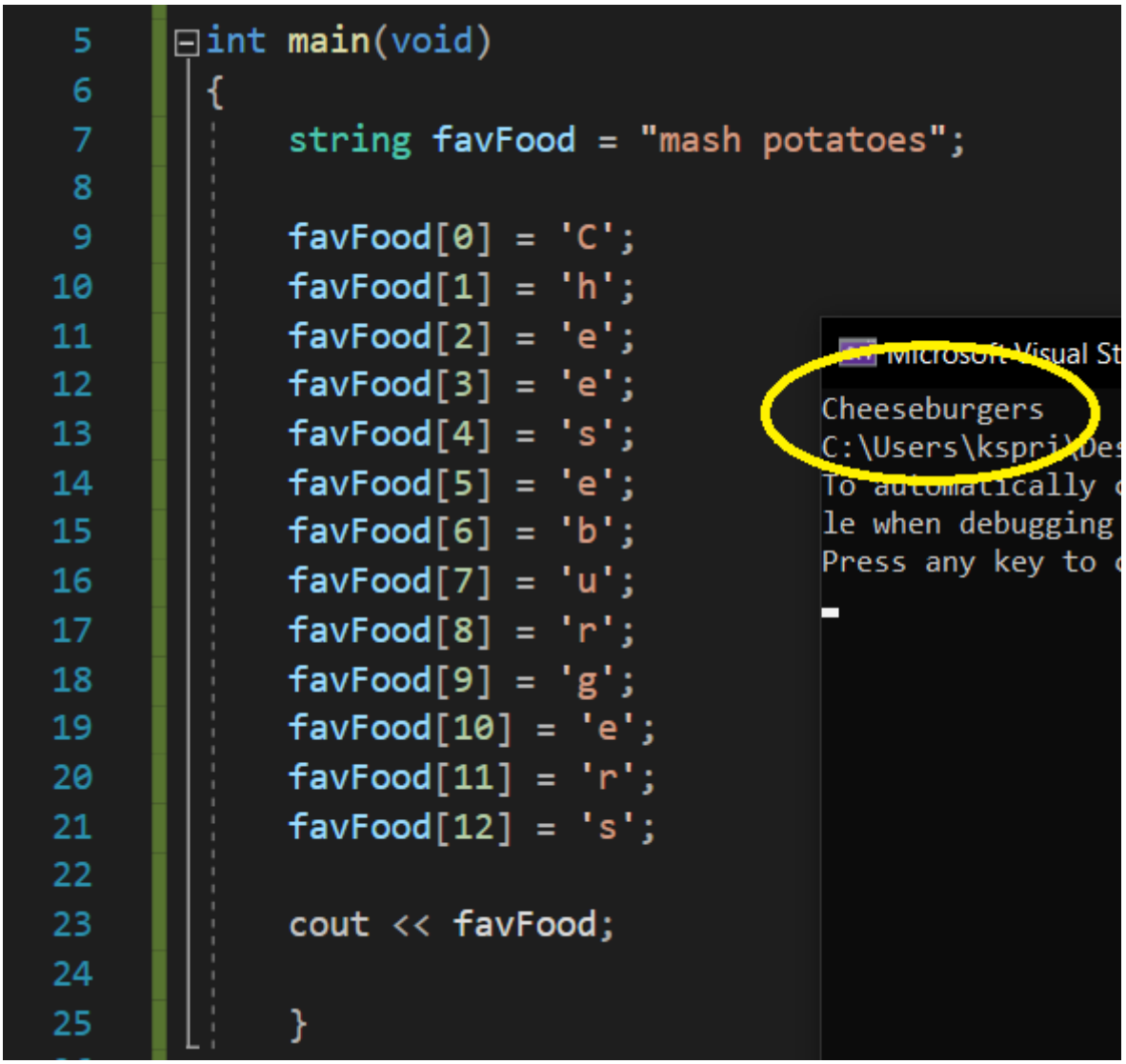
Activity #3 - Change Characters In A String

Create a string variable that stores the value “mash potatoes”

Change every character in the string, one by one to make it “cheeseburgers” then display your variable

_____

Do not look at the next page until you have completed the activity!



MATH

We are going to include another library in our program. Remember where we include our libraries that include pre-built functions? If you said in the header or at the top then you are correct! Go ahead and place the following include statement in the header of your program:

#include

```
1  #include<iostream>
2  #include <string>
3  #include <cmath>
4  using namespace std;
```

Math functions can be used a lot in programming, especially if you are going to create any type of visual or game based application! Luckily you do not need to be a mathematician here. Let's look into a couple Math functions that are included within our math library. Since you are already familiar with functions, these should be very easy for you!

max() Function

The max() function takes two integers as parameters and will find the highest value. So for example, let's see what happens when we pass in 5 and 832:

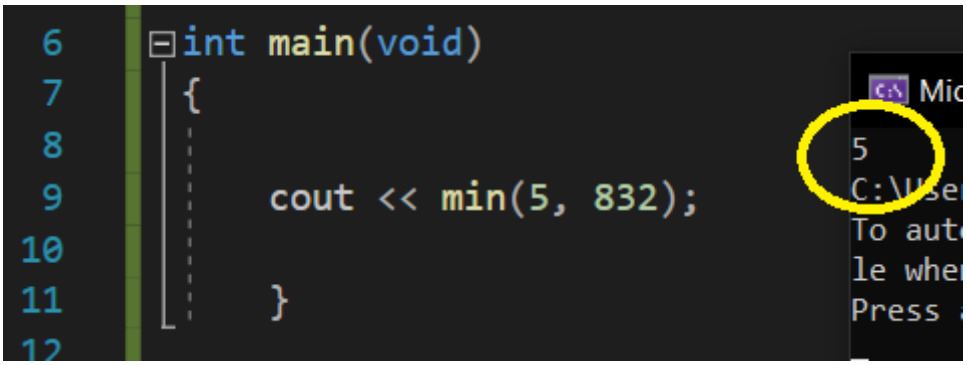
```
6  int main(void)
7  {
8
9      cout << max(5, 832);
10
11     }
```

```
832
To aut
le whe
```

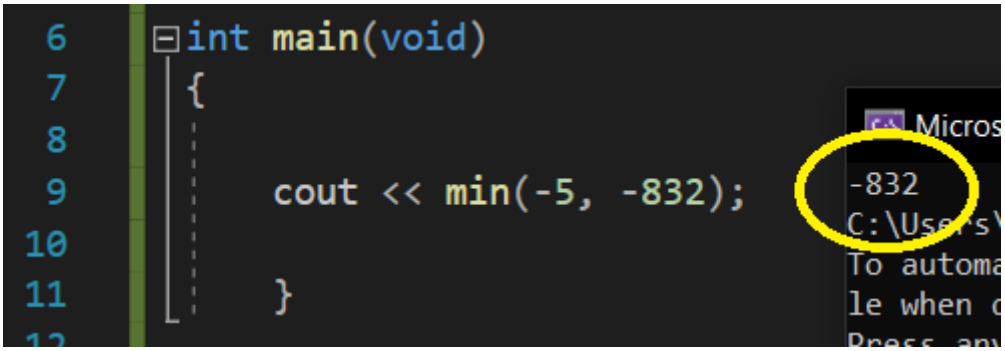
Yes, it is that simple! Don't be fooled though... some of these simple Math functions will be very useful to you as we progress!

min() Function

The min() function finds the lowest value between the two integers we pass to it. Let's try the same numbers as we did with the max() function:

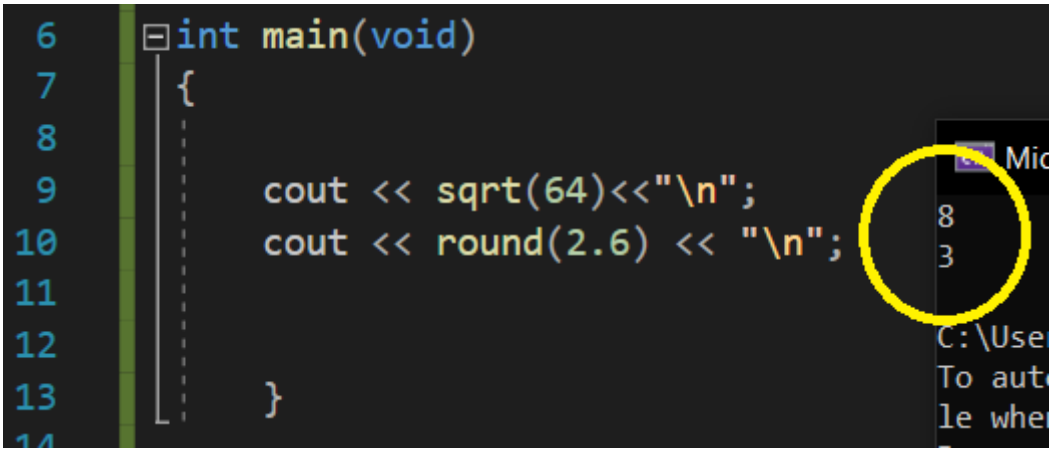


You can also do this with negative integers:



sqrt() Function

This function gives the square root of any integer you pass in



round() Function

And lastly, the round() function rounds your floating decimal number to the nearest whole integer.

ARRAYS

In a previous chapter you learned how to access actual characters in a string and even change those characters to make completely new strings! Now that you understand how this is possible, we will easily step into the world of arrays. Arrays are highly used in any programming language and you don't realize it yet... but you basically already know how! Pretty cool huh?

Arrays are a way to store multiple values instead of storing those values in their own variables! The way we write an array is simple. You give your array a data type, name it, tell it how many items are in it (within brackets [ ]), then fill your array inside braces { }.

```
string favoriteFood [5] = {"Pizza","Calzone","BLT","Taco","Ravioli"};
```

So instead of making five different variables to hold these values, we can stick them in one array variable. You can also do this with integers by giving your array an int data type.

```
int favoriteNumbers[7] = {77,24,12,69,17,10,143};
```

As you can see, strings always have the quotation marks around them and integers do not.

C++ also allows us to leave the number of items between our brackets [ ] empty! If you leave it empty, the compiler will assume the size based on how many values are in the array. This is helpful if you plan on changing and messing around with your array. I personally always leave it empty unless it is a list that I specifically do not want changed. When you want a list to be permanent and protected against any changes you will make the array constant by typing 'const' before your array:

```
const int favoriteNumbers[7] = {77,24,12,69,17,10,143};
```

Ok, now that you understand an array let's go ahead and create our int and string arrays:

```
6   int main(void)
7   {
8       string favoriteFood[] = { "Pizza","Calzone","BLT","Taco","Ravioli"};
9       int favoriteNumbers[] = { 77,24,12,69,17,10,143 };
10
11
12      }
```

Now, just like we previously accessed each index of our strings we will access the indices of our array. Let's print out our string array, line by line:

```
6   int main(void)
7   {
8       string favoriteFood[] = { "Pizza","Calzone","BLT","Taco","Ravioli"};
9       int favoriteNumbers[] = { 77,24,12,69,17,10,143 };
10
11      cout << favoriteFood[0] << "\n";
12      cout << favoriteFood[1] << "\n";
13      cout << favoriteFood[2] << "\n";
14      cout << favoriteFood[3] << "\n";
15      cout << favoriteFood[4] << "\n";
16
17      }
```

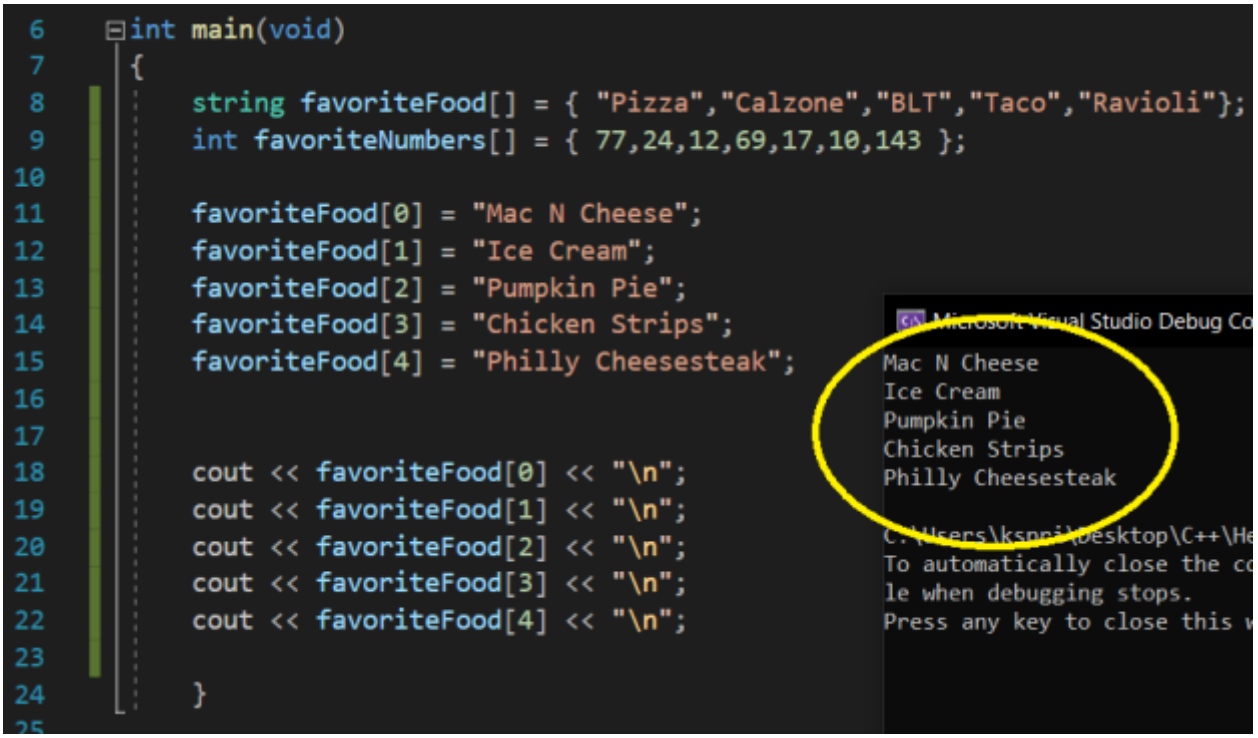
```
Microsoft Visual Studio Debug Console
Pizza
Calzone
BLT
Taco
Ravioli
C:\Users\kspri\Desktop\C++\HelloWo
```

We will try the same thing with our favoriteNumbers array:

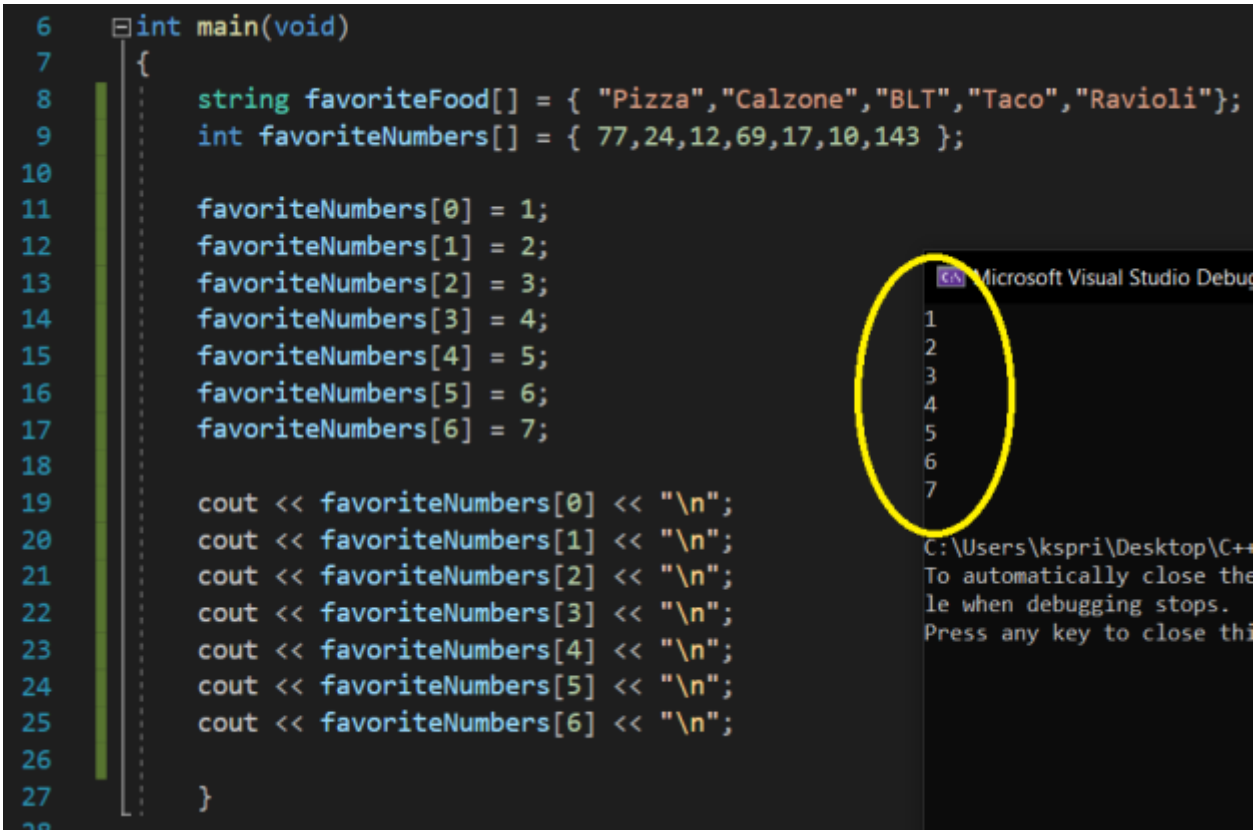
```
6   int main(void)
7   {
8       string favoriteFood[] = { "Pizza","Calzone","BLT","Taco","Ravioli"};
9       int favoriteNumbers[] = { 77,24,12,69,17,10,143 };
10
11      cout << favoriteNumbers[0] << "\n";
12      cout << favoriteNumbers[1] << "\n";
13      cout << favoriteNumbers[2] << "\n";
14      cout << favoriteNumbers[3] << "\n";
15      cout << favoriteNumbers[4] << "\n";
16      cout << favoriteNumbers[5] << "\n";
17      cout << favoriteNumbers[6] << "\n";
18      }
19
```

```
Microsoft Visual Studio Debug
77
24
12
69
17
10
143
```

Next, let's change the elements in our array by stating what index position, then assigning (=) a new value to that index spot. I will change every element in our favoriteFood array then print each element out to see if the array was changed correctly:



Same for our favoriteNumbers array:



Great! After you complete the practice activity below, we will start diving into conditions and if statements!

PRACTICE ACTIVITIES

Activity #1 - Access 2 Arrays To Make Sentences

Using the example we just went through with the favoriteFood and favoriteNumbers arrays:

Go into your favoriteFood array and add 2 new elements so that both arrays are the same length.

Now taking the first element from each array, then the second and so on, print out the following statements, each on a new line:

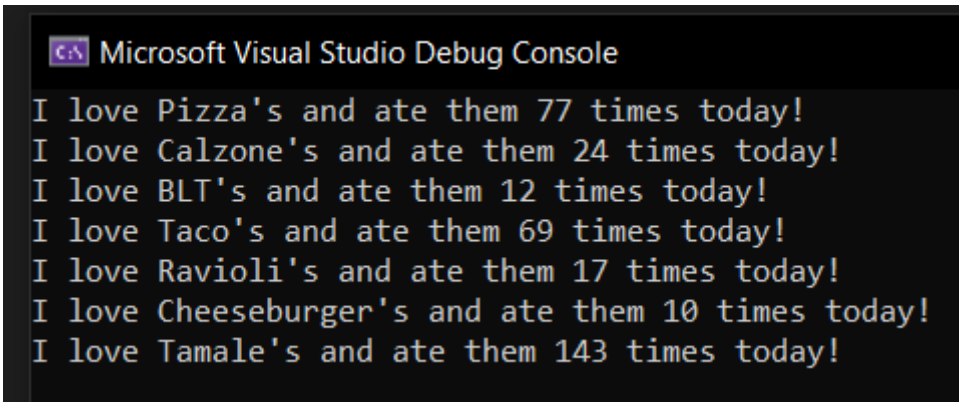
I love—favoriteFood[0]—'s and ate them—favoriteNumbers[0]—times today!

...keep doing this until you have printed every element in your arrays

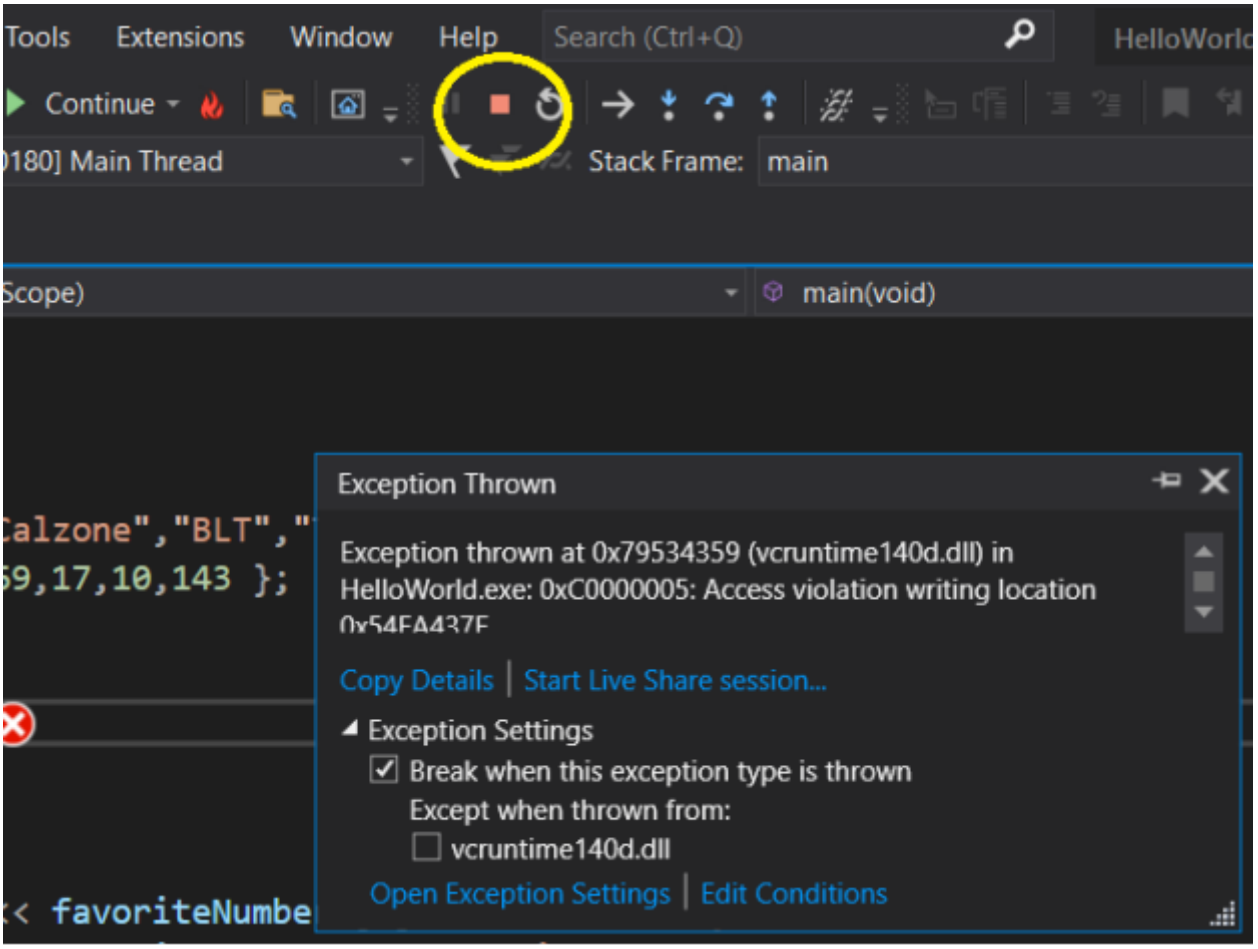
*Hint (remember when you add more than one variable to a string concatenated list you must surround the variables with <<)

Do not look at the next page until you have completed the activity!

```
int main(void)
{
    string favoriteFood[] = { "Pizza","Calzone","BLT","Taco","Ravioli","Cheeseburger","Tamale"};
    int favoriteNumbers[] = { 77,24,12,69,17,10,143 };


    cout << "I love " + favoriteFood[0] + "'s and ate them " << favoriteNumbers[0] << " times today!\n";
    cout << "I love " + favoriteFood[1] + "'s and ate them " << favoriteNumbers[1] << " times today!\n";
    cout << "I love " + favoriteFood[2] + "'s and ate them " << favoriteNumbers[2] << " times today!\n";
    cout << "I love " + favoriteFood[3] + "'s and ate them " << favoriteNumbers[3] << " times today!\n";
    cout << "I love " + favoriteFood[4] + "'s and ate them " << favoriteNumbers[4] << " times today!\n";
    cout << "I love " + favoriteFood[5] + "'s and ate them " << favoriteNumbers[5] << " times today!\n";
    cout << "I love " + favoriteFood[6] + "'s and ate them " << favoriteNumbers[6] << " times today!\n";

    }
```


Microsoft Visual Studio Debug Console
I love Pizza's and ate them 77 times today!
I love Calzone's and ate them 24 times today!
I love BLT's and ate them 12 times today!
I love Taco's and ate them 69 times today!
I love Ravioli's and ate them 17 times today!
I love Cheeseburger's and ate them 10 times today!
I love Tamale's and ate them 143 times today!

Help - If you ever run your program and an error occurs simply click on the red stop button. We will learn how to handle thrown exceptions later on.



CONDITIONS AND IF STATEMENTS

Decision-making makes our C++ programs more flexible, interesting and smart. Just as we make decisions in our life, we can make decisions in code by using if statements. An if statement is a block of code that allows you to control the path the computer will take when it executes your code. This is important because when we write more complex and longer programs we don't really want the computer to run all of our code. We only want to run certain parts of our code

when it makes sense or when we decide it is the right time to do so. if statements give us this decision-making ability.

How is that? If statements allow us to set up a condition that needs to be met before any additional code is executed. This condition is usually a Boolean expression, which is a condition that the computer evaluates and decides is either True or False. Think of Boolean expressions as "Yes or No" questions, where a "Yes is True and a 'No" is False. Our usual logical conditions are:

Less than: a < b

Less than or equal to: a <= b

Greater than: a> b

Greater than or equal to: a >= b

Equal to: a == b

Not Equal to: a != b

You can literally use these conditions to perform different actions based on whether or not the condition is true or false.

Here's how to write an it statement

string mood=”hungry”

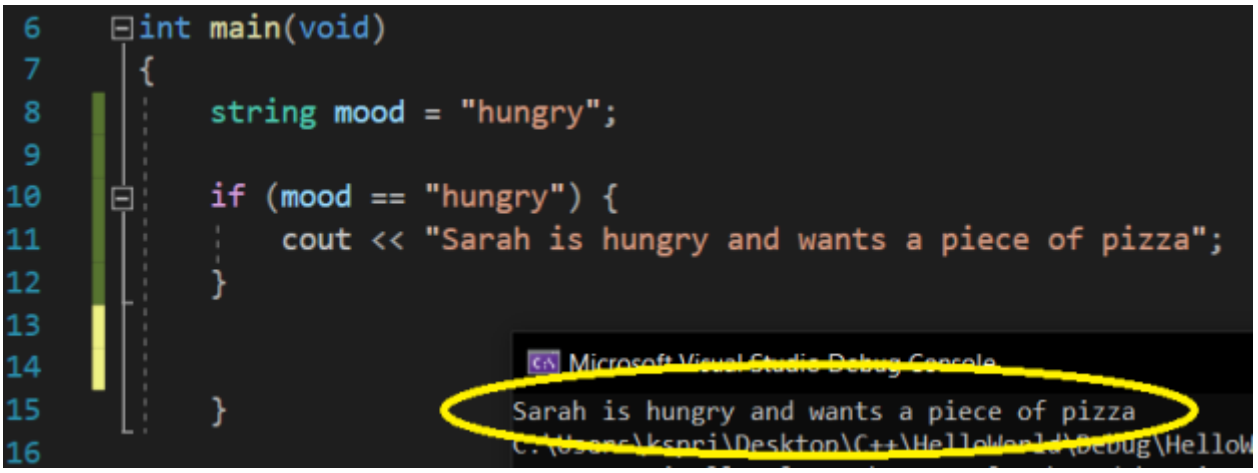
if (mood == “hungry”){

cout<<”Sarah is hungry and wants a piece of pizza”;

}

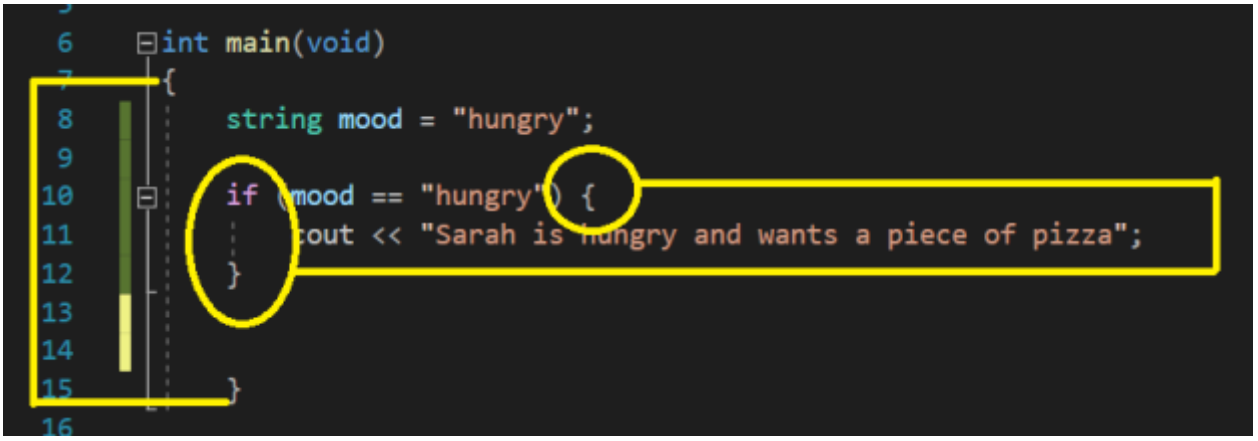
Pretty logical, right? Code always runs from top to bottom so let’s analyze exactly what is happening here. The if statement starts off analyzing a condition, which is whatever you place in the parenthesis. The answer is a true or false boolean expression. In this case we are asking if our mood variable is equal to this “hungry” string...

IF it is True that mood is equal to hungry then the computer will execute the next line of code located WITHIN the braces. IF it is False that mood is equal to hungry then it skips the next lines of code that are indented within the if statement actions and finds the next block of code.


int main(void)
{
string mood = "hungry";
if (mood == "hungry") {
cout << "Sarah is hungry and wants a piece of pizza";
}
}
Microsoft Visual Studio Debug Console
Sarah is hungry and wants a piece of pizza

Because all programs run top to bottom, the mood variable was set, then the if conditional question was true, so it ran the if statement’s code!

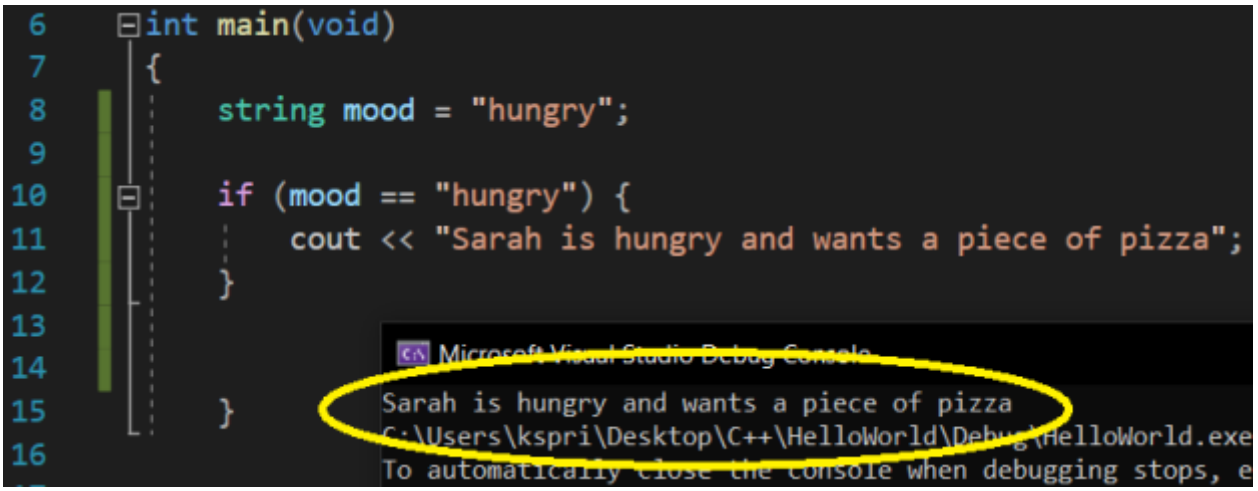
Notice how the if statement has its own curly braces… we are only used to seeing that at the beginning and end of our main! This shows us that our if statement is almost like its own little program where all kinds of things can happen inside the curly braces! The indention here is important and you need to always follow the rules of lining up your curly braces! Everything that happens inside these curly braces is in the scope of our if statement.



Let's practice! Code what I just showed you but set our mood variable to "happy". Then type out the if statement and run your program:

Nothing happened, did it? As you can see, nothing happened because mood was not equal to hungry so the code inside the if

statement did not run. Now try it again but assign “hungry” to the mood variable.


int main(void)
{
string mood = "hungry";
if (mood == "hungry") {
cout << "Sarah is hungry and wants a piece of pizza";
}
}
Sarah is hungry and wants a piece of pizza
C:\Users\kspri\Desktop\C++\HelloWorld\Debug\HelloWorld.exe
To automatically close the console when debugging stops, e

Wonderfully done! Now that our if statement is true, the code runs within the if statement! Within this block of code we have the print () method to print out the string about sarah eating a slice of pizza. Depending on what you are doing you could have as many code blocks and even other functions within an if statement!

What if we are not hungry though? We can add that decision into our code, too, using an else statement. The else statement just says, hey, that IF conditional question was answered false so I will run my code. When the IF conditional question is true, the else statement will not run.

The code would look like this:

```
6   int main(void)
7   {
8       string mood = "hungry";
9
10      if (mood == "hungry") {
11          cout << "Sarah is hungry and wants a piece of pizza";
12      }
13      else {
14          cout << "Sarah is not hungry!";
15      }
16
17
18      }
```

Did you notice that the else statement also has its own curly braces to enclose its code? Try running your program now:

```
6   int main(void)
7   {
8       string mood = "hungry";
9
10      if (mood == "hungry") {
11          cout << "Sarah is hungry and wants a piece of pizza";
12      }
13      else {
14          cout << "Sarah is not hungry!";
15      }
16
17
18      }
19
```

Microsoft Visual Studio Debug Console

Sarah is hungry and wants a piece of pizza

C:\Users\kspri\Desktop\C++\HelloWorld\Debug\Hello

To automatically close the console when debugging

Our code ran what it should have because mood is equal to hungry. Please note that if we had named our mood variable Hungry with a capital H, it would have not been equal to hungry. Now go change the mood variable to Super Hungry and run your program:

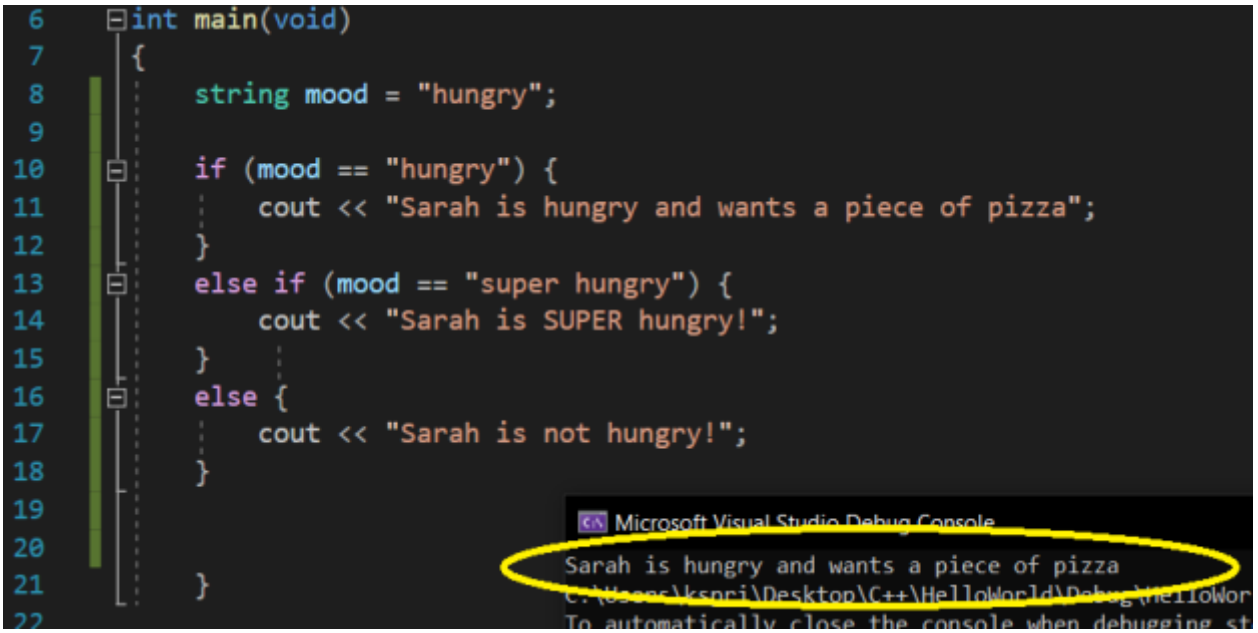
```
6   int main(void)
7   {
8       string mood = "Super Hungry";
9
10      if (mood == "hungry") {
11          cout << "Sarah is hungry and wants a piece of pizza";
12      }
13      else {
14          cout << "Sarah is not hungry!";
15      }
16
17
18      }
19
```

```
Sarah is not hungry!
C:\Users\kspri\Desktop\
To automatically close t
le when debugging stops.
Press any key to close t
```

Now, what if we wanted more than one conditional question for our if statement? What if we also wanted to test and see if mood is equal to something else? This is where the if else statement comes in! It is just like the if statement but it runs after the 1st conditional question in the main if statement. Let's add our else if statement with our 2nd conditional question. Take a look at this code and copy it into your program:

```
6   int main(void)
7   {
8       string mood = "Hungry";
9
10      if (mood == "hungry") {
11          cout << "Sarah is hungry and wants a piece of pizza";
12      }
13      else if (mood == "super hungry") {
14          cout << "Sarah is SUPER hungry!";
15      }
16      else {
17          cout << "Sarah is not hungry!";
18      }
19
20
21      }
```

Our code is running from top to bottom and is saying, hey if mood is hungry I will print this out. If mood is not hungry it will ask if mood is super hungry and if it is it will print this sentence out. Then if mood was never hungry or super hungry I will print out this else block of code. Go ahead and change your mood variable to each one and see how our program is now running with more logic than ever before!


int main(void)
{
string mood = "hungry";
if (mood == "hungry") {
cout << "Sarah is hungry and wants a piece of pizza";
}
else if (mood == "super hungry") {
cout << "Sarah is SUPER hungry!";
}
else {
cout << "Sarah is not hungry!";
}
}
Microsoft Visual Studio Debug Console
Sarah is hungry and wants a piece of pizza

```
6   int main(void)
7   {
8       string mood = "super hungry";
9
10      if (mood == "hungry") {
11          cout << "Sarah is hungry and wants a piece of pizza";
12      }
13      else if (mood == "super hungry") {
14          cout << "Sarah is SUPER hungry!";
15      }
16      else {
17          cout << "Sarah is not hungry!";
18      }
19
20
21      }
22
```

```
Microsoft Visual Studio Debug Console
Sarah is SUPER hungry!
C:\Users\kspri\Desktop\C++\HelloWorl
To automatically close the console w
```

```
6   int main(void)
7   {
8       string mood = "kind of hungry";
9
10      if (mood == "hungry") {
11          cout << "Sarah is hungry and wants a piece of pizza";
12      }
13      else if (mood == "super hungry") {
14          cout << "Sarah is SUPER hungry!";
15      }
16      else {
17          cout << "Sarah is not hungry!";
18      }
19
20
21      }
22
```

```
Sarah is not hungry!
To automatically close the co
```

Now that you understand how if-else if-else statements work let's create a little guessing game based on user input together! This

game is going to be coded with a lot of if statements. First let's set the rules as a guideline, then we can logically code it:

- The user must guess a number between 1-10
- There is only one correct number
- If the user guesses the correct number, they win
- For each wrong number guessed we will let them know the game is over and tell them how many digits away from the correct answer they were

Wait, hold on a minute! Are you thinking what I'm thinking? Our user input is a string, so how are we supposed to compare a string to our winning integer number?! Luckily, there is a string stream we can include in our header to help with that!

First, place the #include into your header, then as we create this guessing game I will show you exactly how to convert a string to integer:

```
1  #include<iostream>
2  #include <string>
3  #include <cmath>
4  #include <sstream>
5  using namespace std;
6
```

Ok! Let's start with setting a variable to hold our winning number.

```
7   int main(void)
8   {
9       int winningNum = 7;
10
11
12      }
13
```

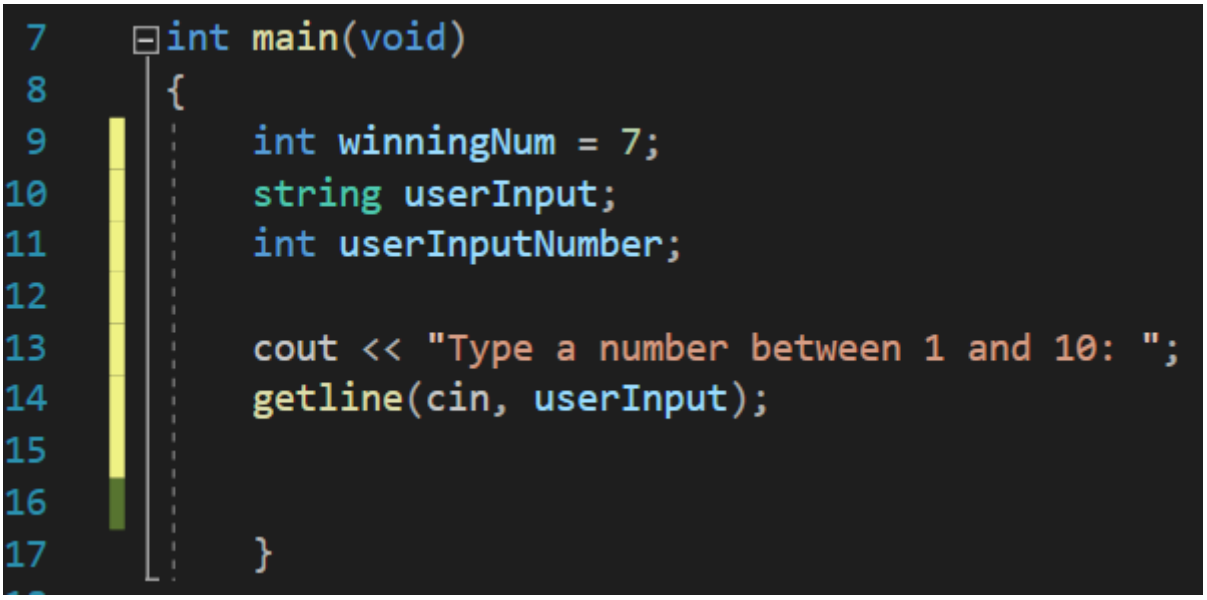
Let's set an empty string variable where we can store the user input:

```
7   int main(void)
8   {
9       int winningNum = 7;
10      string userInput;
11
12      }
13
```

Now let's get our first number from the user and store it in the userInput variable:

```
7   int main(void)
8   {
9       int winningNum = 7;
10      string userInput;
11
12      cout << "Type a number between 1 and 10: ";
13      getline(cin, userInput);
14
15
16      }
```

Before we convert the userInput string to an integer we will need to create a variable to hold that converted integer:



With 2 statements we can convert userInput to userInputNumber:

stringstream intValue(userInput);

intValue >> userInputNumber;

The stringstream is simply declaring the function it is about to use. The function is

intValue(—place your string variable here—)

This is the function that takes your string through a portal, transforms it into an integer, then stores it in the intValue.

Now all we need to do is take that integer sitting in the intValue and assign it to our userInputNumber variable!

```
 7  int main(void)
 8  {
 9      int winningNum = 7;
10      string userInput;
11      int userInputNumber;
12
13      cout << "Type a number between 1 and 10: ";
14      getline(cin, userInput);
15
16      stringstream intValue(userInput);
17      intValue >> userInputNumber;
18
19      }
```

Let's test and print to see if this is working correctly by doing a simple math equation that could not be accomplished with a string and integer.

```
 7  int main(void)
 8  {
 9      int winningNum = 7;
10      string userInput;
11      int userInputNumber;
12
13      cout << "Type a number between 1 and 10: ";
14      getline(cin, userInput);
15
16      stringstream intValue(userInput);
17      intValue >> userInputNumber;
18
19      cout << userInputNumber * 1;
20
21      }
```

```
Microsoft Visu
Type a number
8
C:\Users\kspri
To automatical
```

Great! We are good to go and now you have a powerful little tool to convert user input or other strings to integers in the

future!

Ok, we know that if the user guesses the correct number, they win - so let's make that our first if statement conditional question! Be creative, but follow along with me for guidance:

```
7   int main(void)
8   {
9       int winningNum = 7;
10      string userInput;
11      int userInputNumber;
12
13      cout << "Type a number between 1 and 10: ";
14      getline(cin, userInput);
15
16      stringstream intValue(userInput);
17      intValue >> userInputNumber;
18
19      if (userInputNumber == winningNum) {
20          cout << "*****CONGRATULATIONS!******  You guessed the winning number "<<userInputNumber;
21      }
22
23
24
25      }
```

Next, let's add our first else if statement if the user types one and that is not the winning number. Within this output we want to tell the user how many digits off from the winning number they were. To do this, we will need to find what number is lowest and subtract it from the other number... do you remember what function allows you to do this? Hint... min(x,x) and max(x,x)! We need to do this because you do not want your calculation to produce a negative number.

```
int main(void)
{
    int winningNum = 7;
    string userInput;
    int userInputNumber;

    cout << "Type a number between 1 and 10: ";
    getline(cin, userInput);

    stringstream intValue(userInput);
    intValue >> userInputNumber;

    if (userInputNumber == winningNum) {
        cout << "*****CONGRATULATIONS!******  You guessed the winning number "<<userInputNumber;
    }
    else if (userInputNumber == 1) {
        int lowestNum = min(winningNum, 1);
        int highestNum = max(winningNum, 1);
        int difference = highestNum - lowestNum;
        cout << "*****GAME OVER*****  You were " << difference << " digits away from the winning number!";
    }


    }
```

Now you can code a new else if statement for every number 1 through 10. Remember, none of these codes will run if the one before it runs. Ok, let's see the finished product:

```
7   int main(void)
8   {
9       int winningNum = 7;
10      string userInput;
11      int userInputNumber;
12
13      cout << "Type a number between 1 and 10: ";
14      getline(cin, userInput);
15
16      stringstream intValue(userInput);
17      intValue >> userInputNumber;
18
19      if (userInputNumber == winningNum) {
20          cout << "*****CONGRATULATIONS!******  You guessed the winning number "<<userInputNumber;
21      }
22      else if (userInputNumber == 1) {
23          int lowestNum = min(winningNum, 1);
24          int highestNum = max(winningNum, 1);
25          int difference = highestNum - lowestNum;
26          cout << "*****GAME OVER*****  You were " << difference << " digits away from the winning number!";
27      }
28      else if (userInputNumber == 2) {
29          int lowestNum = min(winningNum, 2);
30          int highestNum = max(winningNum, 2);
31          int difference = highestNum - lowestNum;
32          cout << "*****GAME OVER*****  You were " << difference << " digits away from the winning number!";
33      }
```

```
34      else if (userInputNumber == 3) {
35          int lowestNum = min(winningNum, 3);
36          int highestNum = max(winningNum, 3);
37          int difference = highestNum - lowestNum;
38          cout << "*****GAME OVER*****  You were " << difference << " digits away from the winning number!";
39      }
40      else if (userInputNumber == 4) {
41          int lowestNum = min(winningNum, 4);
42          int highestNum = max(winningNum, 4);
43          int difference = highestNum - lowestNum;
44          cout << "*****GAME OVER*****  You were " << difference << " digits away from the winning number!";
45      }
46      else if (userInputNumber == 5) {
47          int lowestNum = min(winningNum, 5);
48          int highestNum = max(winningNum, 5);
49          int difference = highestNum - lowestNum;
50          cout << "*****GAME OVER*****  You were " << difference << " digits away from the winning number!";
51      }
52      else if (userInputNumber == 6) {
53          int lowestNum = min(winningNum, 6);
54          int highestNum = max(winningNum, 6);
55          int difference = highestNum - lowestNum;
56          cout << "*****GAME OVER*****  You were " << difference << " digits away from the winning number!";
57      }
58      else if (userInputNumber == 7) {
59          int lowestNum = min(winningNum, 7);
60          int highestNum = max(winningNum, 7);
61          int difference = highestNum - lowestNum;
62          cout << "*****GAME OVER*****  You were " << difference << " digits away from the winning number!";
63      }
```

```
64      else if (userInputNumber == 8) {
65          int lowestNum = min(winningNum, 8);
66          int highestNum = max(winningNum, 8);
67          int difference = highestNum - lowestNum;
68          cout << "*****GAME OVER*****  You were " << difference << " digits away from the winning number!";
69      }
70      else if (userInputNumber == 9) {
71          int lowestNum = min(winningNum, 9);
72          int highestNum = max(winningNum, 9);
73          int difference = highestNum - lowestNum;
74          cout << "*****GAME OVER*****  You were " << difference << " digits away from the winning number!";
75      }
76      else if (userInputNumber == 10) {
77          int lowestNum = min(winningNum, 10);
78          int highestNum = max(winningNum, 10);
79          int difference = highestNum - lowestNum;
80          cout << "*****GAME OVER*****  You were " << difference << " digits away from the winning number!";
81      }
82
83      }
```

Whew, that seems like a lot of code! Are you starting to fully understand how if-else statements work?

PRACTICE ACTIVITY

(If-Else Statements)

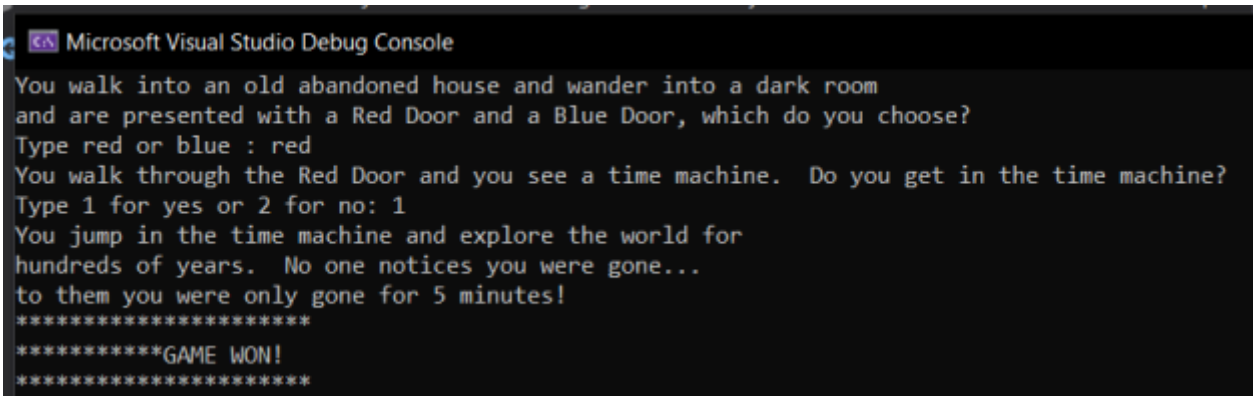
Activity #1 - Secret Door Game

We want the user to choose between 2 different colored doors. Within each door will be 2 choices (choice #1 or #2). One choice will be something bad happening and game over and one choice will be something awesome happening and game won. Use your creativity!

Tip: Since you are not performing math operations, you can see if 2 strings are equal in your if statements, you will not need to convert any strings to integers.

Do not look at the next page until you have completed the activity!

Outputs:


Microsoft Visual Studio Debug Console
You walk into an old abandoned house and wander into a dark room
and are presented with a Red Door and a Blue Door, which do you choose?
Type red or blue : red
You walk through the Red Door and you see a time machine. Do you get in the time machine?
Type 1 for yes or 2 for no: 1
You jump in the time machine and explore the world for
hundreds of years. No one notices you were gone...
to them you were only gone for 5 minutes!
****
**********GAME WON!
****

```
You walk into an old abandoned house and wander into a dark room
and are presented with a Red Door and a Blue Door, which do you choose?
Type red or blue : red
You walk through the Red Door and you see a time machine.  Do you get in the time machine?
Type 1 for yes or 2 for no: 2
You turn to walk away and the time machine starts glitching
it starts sputtering, pulls you in and sends you back in time 800 years!  You are stuck there...
**********
**********GAME OVER!
**********
```

```
Microsoft Visual Studio Debug Console
You walk into an old abandoned house and wander into a dark room
and are presented with a Red Door and a Blue Door, which do you choose?
Type red or blue : blue
You slowly through the Blue Door and a princess is standing there.
She asks you to help her. Do you help? Type 1 for yes or 2 for no: 1
You help the princess slay a dragon which frees an enchanted prince.
You three go on to save the entire realm!
**********
**********GAME WON!
**********
```

```
Microsoft Visual Studio Debug Console
You walk into an old abandoned house and wander into a dark room
and are presented with a Red Door and a Blue Door, which do you choose?
Type red or blue : blue
You slowly through the Blue Door and a princess is standing there.
She asks you to help her. Do you help? Type 1 for yes or 2 for no: 2
You start to leave and a dragon swoops down and flies off with the princess... never to be seen again
**********
**********GAME OVER!
**********
```

```
7   int main(void)
8   {
9       string userDoorChoice;
10      string userStringOneOrTwoChoice;
11
12      cout << "You walk into an old abandoned house and wander into a
13      getline(cin, userDoorChoice);
14
15
16      if (userDoorChoice == "red") {
17          cout << "You walk through the Red Door and you see a time ma
18          getline(cin, userStringOneOrTwoChoice);
19
20          if (userStringOneOrTwoChoice == "1") {
21              cout << "You jump in the time machine and explore the wo
22              }
23
24          else if (userStringOneOrTwoChoice == "2") {
25              cout << "You turn to walk away and the time machine star
26
27          }
28      }
29
```

```
30
31      else if (userDoorChoice == "blue") {
32          cout << "You slowly through the Blue Door and a princess is
33          getline(cin, userStringOneOrTwoChoice);
34
35          if (userStringOneOrTwoChoice == "1") {
36              cout << "You help the princess slay a dragon which frees
37          }
38
39          else if (userStringOneOrTwoChoice == "2") {
40              cout << "You start to leave and a dragon swoops down and
41
42          }
43
44      }
```

FOR LOOPS

What is a loop you ask? Remember the Choose A Door game we just created? What if at the end of the game we could let the user decide if they wanted to replay and start the game over? They could play the game a million times or whenever they choose to quit or stop the game. This is a very simple example of how a loop works. Understanding loops will enable you to create very powerful programs.

A big part of why computers are so powerful is that they can repeat many actions or calculations very quickly. One of the ways we can tell a computer to do this is through for loops or while loops. A loop is a special kind of programming statement that allows you to repeat a block of code.

The first kind of loop is called a for loop. This kind of loop repeats a block of code a specific number of times. Think of the for loop like a sentence "For this number of times, go through this thing and execute whatever code I put in this loop, then stop, exit the loop and move on!"

The for loop is written like this:

```
for (statement 1; statement 2; statement 3;){
}
```

Notice that there is a semicolon (;) separating each of these statements. Let's get familiar with what needs to go in each statement section:

Statement 1 - This statements sets a variable before the loop starts and usually looks like this:

```
int i = 0;
```

The ‘int’ is initializing our integer variable ‘i’. You can name your variable anything, but most programmers just name it ‘i” because it stands for iteration, and each time a loop runs it is called an ‘iteration’.

Statement 2 - This statement sets the actual condition (or rules) for the loop to run and usually looks like this:

i < 5;

This conditional statement says ‘hey loop, you will keep looping, over and over again until our variable ‘i’ is greater than 5”. It is saying that as long as i is less than 5 we will oop.

Statement 3 - This statement does whatever you want at the end of EVERY loop (usually you want to increase your ‘i’ counter or decrease it). This usually looks like this:

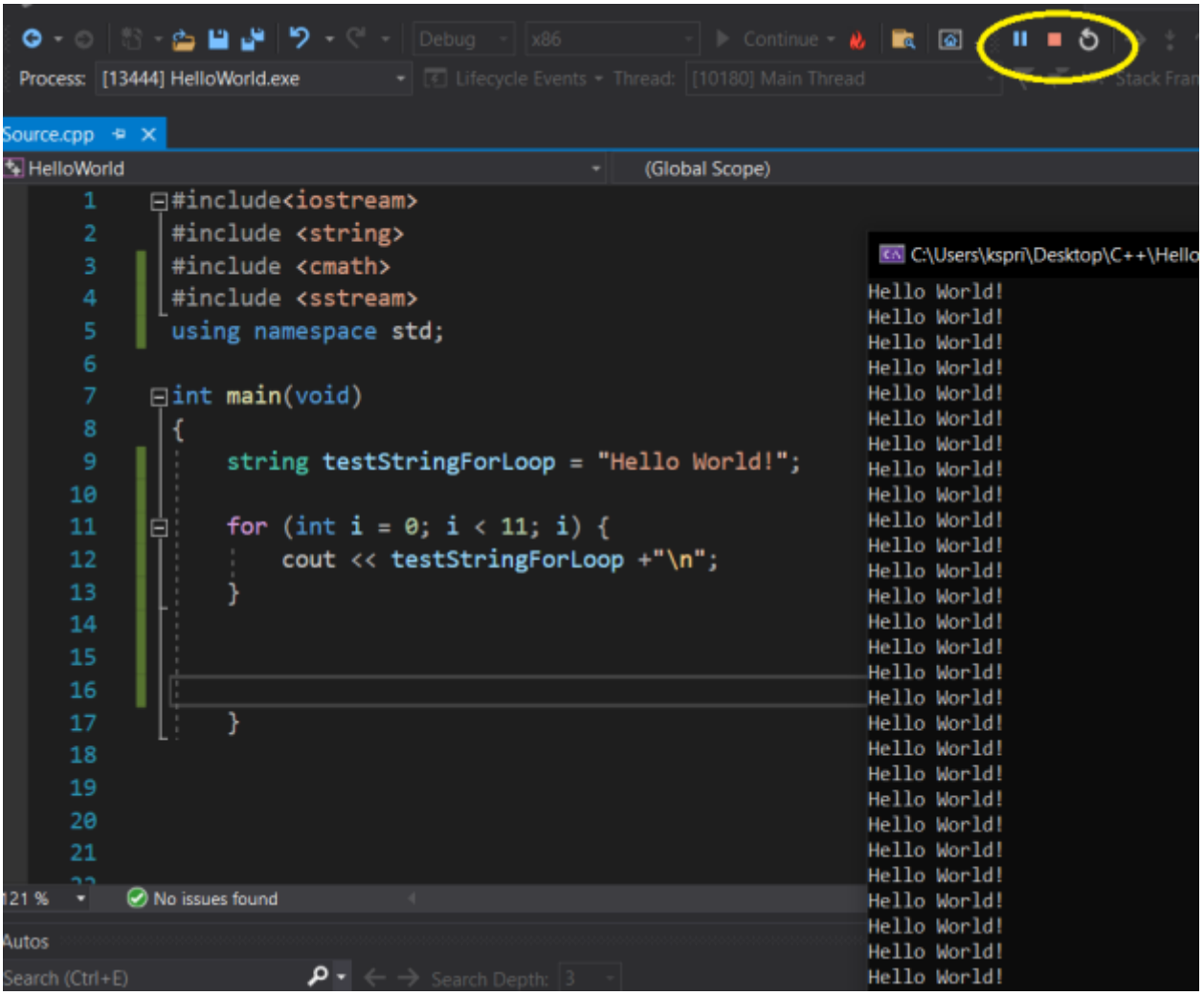
i++;

The i ++ is shorthand for the equation i = i + 1. You could also minus your counter with i—. You can pretty much do whatever you want in this statement depending on what you are trying to accomplish with your loop!

After the for loop statements there are our curly braces again! This of course, means that all code within these braces are in it’s own little world.

Follow along and let’s display the string “Hello!”10 times:

Now delete the ++ after the i in your 3rd statement and run your program:



Congratulations, you have just created an infinite loop! To stop this simply click on the red stop button at the top of your IDE. You never want to create an infinite loop!

Why did this happen though? When the program entered the loop it saw that our 'i' variable was 0. At the end of the loop it was still 0, we never increased the number! So when the loop looks at our conditional statement (i<11) it will never be a false statement because our i variable will always be 0. The heart of understanding loops is understanding this type of programming logic.

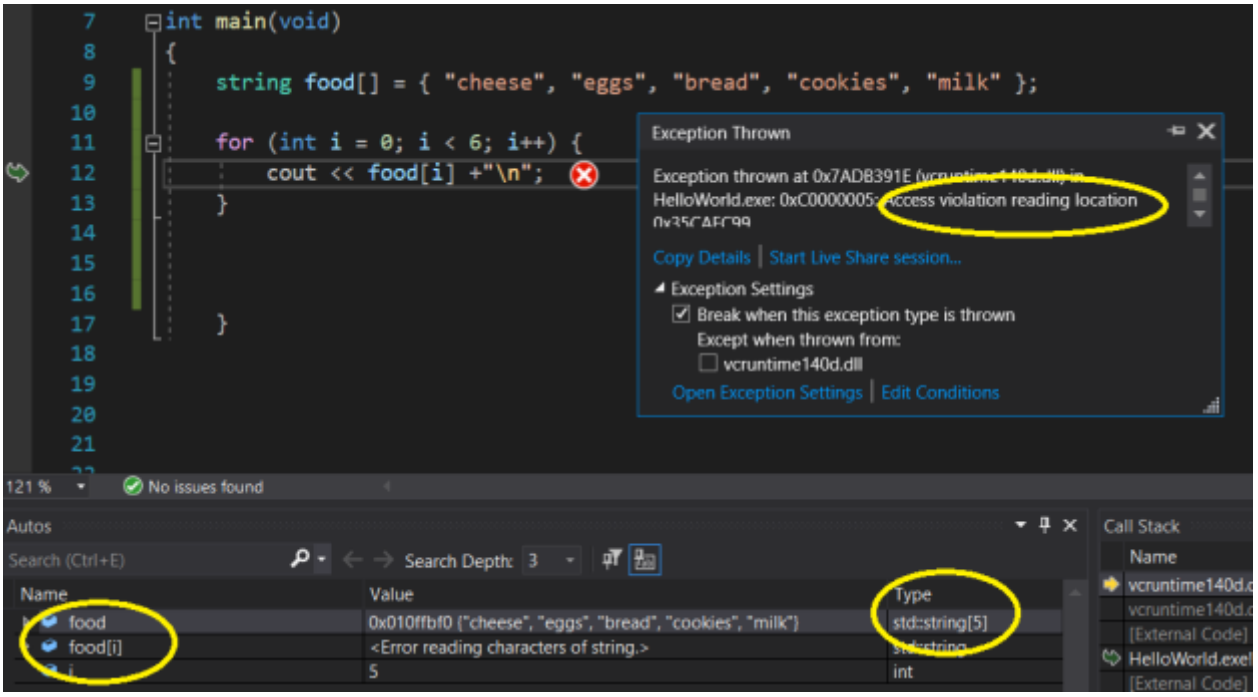
Let's say we create an array of numbers and we want to print out each array element without typing each element out such as

item[1], item [2], etc. Go create a string array named food and place 5 items in it. Now try to create a loop that prints each element on a new line. Remember an array is written like this:

string arrayName[] = { “item”, “item”};

When you ran your code did you have a good outcome? If not, let me explain what is happening in the code pictured above, top to bottom.

We have our array with 5 items of course, however, do not forget that the computer looks at the indices (number of index’s) in our array as 0,1,2,3,4. Next, we start our loop with our counter set at 0. Our second statement is where a lot of students can get tripped up. We only want the loop to run while it’s less than 5 because we are starting at 0. So we think of 4 but it is really 5! Just keep that in mind when dealing with indices… always subtract one and you will be good! If you were to write i < 6 (thinking that 5 is the amount of indices) you would get a thrown exception error and your program would crash:



The error is literally telling us that in our food array, specifically on index location 5, there is a violation with the reading location. Obviously, we know why the computer can't read it, because index #5 does not exist.

As usual with other code functions, anything within our for loop braces can contain whatever code we want.

Let's create a small program together! Follow the guidelines below and try it on your own before moving on.

Our program specs:

We are a small brick-over style pizza company and we want a user to visit our app and order 1 pizza with 3 toppings. At the end of the 1 pizza with 3 toppings order we want to display the order and estimated arrival time.

There are so many ways we could code this program, how are you going to do it?!

Let's start with creating a string array that will hold the 3 toppings our user chooses:

```
string finalToppings[3] = {};
```

Then we will show a welcome message and instructions to the user:

```
cout<< "***Welcome to Pizza Palace!*** Please choose your 3 toppings! \n";
```

Next, let's make a for statement that loops 3 times, because we need to ask the user and store their choice for each topping. Each choice will be stored in an index spot within our finalToppings array:

```
for (int i = 0; i < 3; i++) {
```

We need to ask the user to choose a topping 3 times and do something with their choice. Notice that the 'i' variable is getting passed into the finalToppings[ ] index array when adding the topping. So the first loop will store the user's first choice in finalToppings[0], the second loop will store the user's 2nd choice in finalToppings[1], and the third loop will store the user's 3rd choice in finalToppings[2]:

```
for (int i = 0; i < 3; i++) {
    cout << "\nCHEESE=1  PEPPERONI=2  SAUSAGE=3  ONION=4  GREEN PEPPER=5 -->  ";
    getline(cin, finalToppings[i]);
```

But how do we know what the user actually chose? The way I asked the question to the user only allows them to type in a number! So if the program were to run now, the user choices would be numbers and they would be stored in the finalToppings array. This is where we need to bring our logical if/else-if statements into play!

```
for (int i = 0; i < 3; i++) {
    cout << "\nCHEESE=1  PEPPERONI=2  SAUSAGE=3  ONION=4  GREEN PEPPER=5 -->  ";
    getline(cin, finalToppings[i]);

    if (finalToppings[i] == "1") {
        finalToppings[i] = "CHEESE";
    }
    else if (finalToppings[i] == "2") {
        finalToppings[i] = "PEPPERONI";
    }
    else if (finalToppings[i] == "3") {
        finalToppings[i] = "SAUSAGE";
    }
    else if (finalToppings[i] == "4") {
        finalToppings[i] = "ONION";
    }
    else if (finalToppings[i] == "5") {
        finalToppings[i] = "GREEN PEPPER";
    }
```

Here we are taking the user's numeric choice and making sure to fill the finalToppings array with the correct string choice. Next we will create an output to show the user what they have ordered and when it will arrive:

```
    cout << "\nYou have ordered 1 pizza with the following 3 toppings:\n";

    for (int i = 0; i < 3; i++) {
        cout<<finalToppings[i] + "\n";
    }

    cout << "\nYour order will arrive in 12 minutes!\n";
```

I am using a second for loop here because it is not redundant. It would however be redundant if we had multiple lines, each printing out an index/pizza topping spot. Imagine a program with 1,000's of elements in an array...whew! Here is what this output looks like, with the entire screenshot code below:

```
***Welcome to Pizza Palace!*** Please choose your 3 toppings!

CHEESE=1  PEPPERONI=2  SAUSAGE=3  ONION=4  GREEN PEPPER=5 -->  4

CHEESE=1  PEPPERONI=2  SAUSAGE=3  ONION=4  GREEN PEPPER=5 -->  2

CHEESE=1  PEPPERONI=2  SAUSAGE=3  ONION=4  GREEN PEPPER=5 -->  1

You have ordered 1 pizza with the following 3 toppings:
ONION
PEPPERONI
CHEESE

Your order will arrive in 12 minutes!
```

```cpp
int main(void)
{
    string finalToppings[3] = {};

    cout<< "***Welcome to Pizza Palace!*** Please choose your 3 toppings! \n";

        for (int i = 0; i < 3; i++) {
            cout << "\nCHEESE=1  PEPPERONI=2  SAUSAGE=3  ONION=4  GREEN PEPPER=5 -->  ";
            getline(cin, finalToppings[i]);

            if (finalToppings[i] == "1") {
                finalToppings[i] = "CHEESE";
            }
            else if (finalToppings[i] == "2") {
                finalToppings[i] = "PEPPERONI";
            }
            else if (finalToppings[i] == "3") {
                finalToppings[i] = "SAUSAGE";
            }
            else if (finalToppings[i] == "4") {
                finalToppings[i] = "ONION";
            }
            else if (finalToppings[i] == "5") {
                finalToppings[i] = "GREEN PEPPER";
            }
    }
```

```
        cout << "\nYou have ordered 1 pizza with the following 3 toppings:\n";

        for (int i = 0; i < 3; i++) {
            cout<<finalToppings[i] + "\n";
        }

        cout << "\nYour order will arrive in 12 minutes!\n";
    }
```

As you can see, programming allows you to have a basic set of structures that you can combine, with unlimited creativity to basically make anything you want.

WHILE LOOP

Yes, there is another loop! Our while loop will loop through a block of code as long as a specified condition is true. We write a while loop like this:

```
while(condition){
}
```

If is basically the same as a for loop, except you initialize your counter before the loop, then when you change your counter it would be done within the loop as opposed to being in the opening statement. For example:

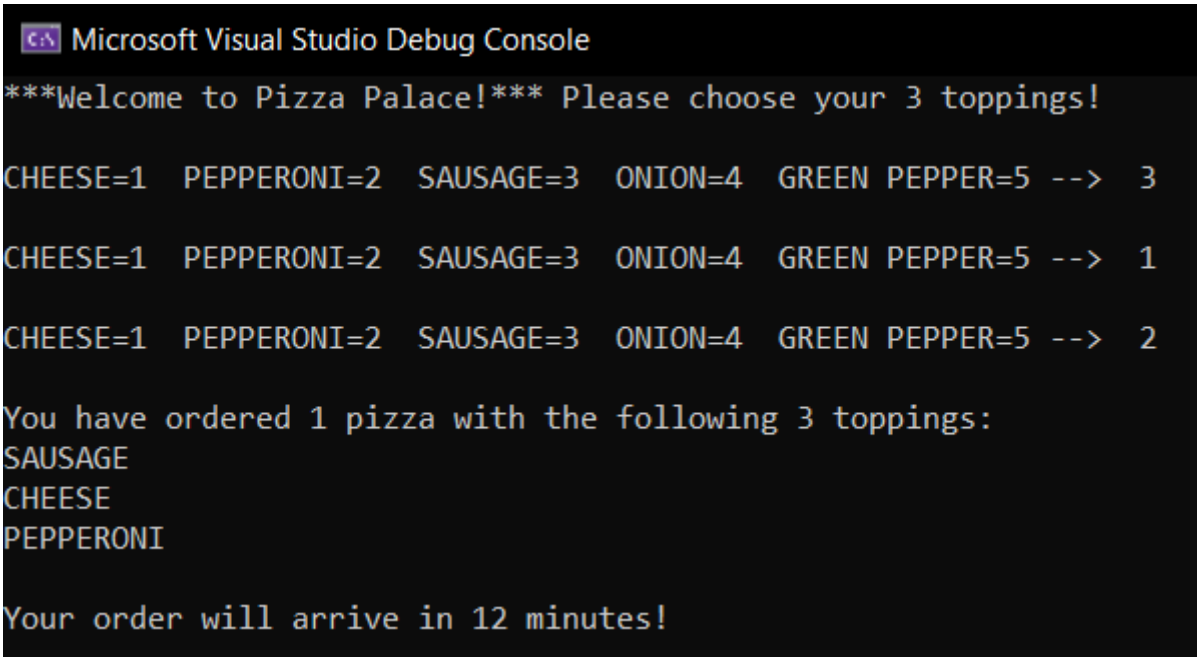
```
int i = 0;
while (i < 5){
cout<<"Hello!";
i++;
}
```

So, you see this is basically the same thing, just written slightly different. Let's change our pizza ordering code to a while loop!

```
12
13      int i = 0;
14      while (i < 3) {
15
16              cout << "\nCHEESE=1  PEPPERONI=2  SAUSAGE=3  ONION=4  GREEN PEPPER=5 -->  ";
17              getline(cin, finalToppings[i]);
18
19              if (finalToppings[i] == "1") {
20                  finalToppings[i] = "CHEESE";
21              }
22              else if (finalToppings[i] == "2") {
23                  finalToppings[i] = "PEPPERONI";
24              }
25              else if (finalToppings[i] == "3") {
26                  finalToppings[i] = "SAUSAGE";
27              }
28              else if (finalToppings[i] == "4") {
29                  finalToppings[i] = "ONION";
30              }
31              else if (finalToppings[i] == "5") {
32                  finalToppings[i] = "GREEN PEPPER";
33              }
34
35              i++;
36      }
```

So, in the for loop we had our statements all together in the beginning ( ) parenthesis. Now in our while loop you can see that we initialized our 'i' counter before ever entering the loop. Then there is only the actual conditional statement in the while ( ) parenthesis. Lastly, after our block(s) of code we increase (or decrease) our counter.

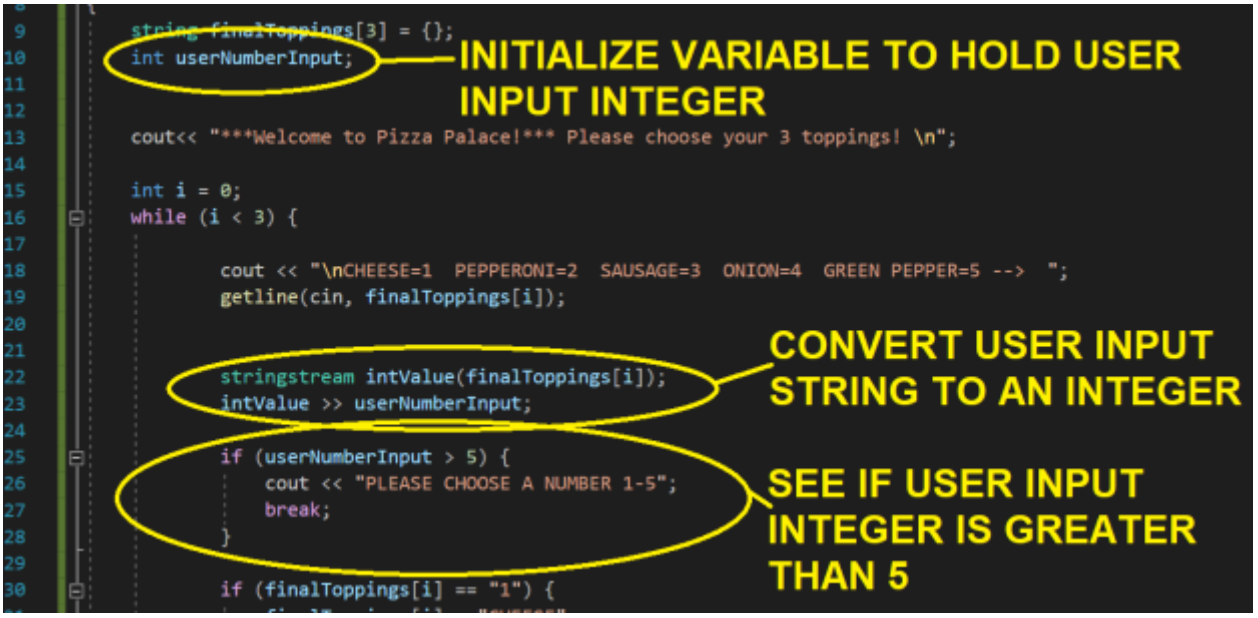
As you can see, when we run our code the output is the same!


Microsoft Visual Studio Debug Console
Welcome to Pizza Palace! Please choose your 3 toppings!
CHEESE=1 PEPPERONI=2 SAUSAGE=3 ONION=4 GREEN PEPPER=5 --> 3
CHEESE=1 PEPPERONI=2 SAUSAGE=3 ONION=4 GREEN PEPPER=5 --> 1
CHEESE=1 PEPPERONI=2 SAUSAGE=3 ONION=4 GREEN PEPPER=5 --> 2
You have ordered 1 pizza with the following 3 toppings:
SAUSAGE
CHEESE
PEPPERONI
Your order will arrive in 12 minutes!

BREAK / CONTINUE

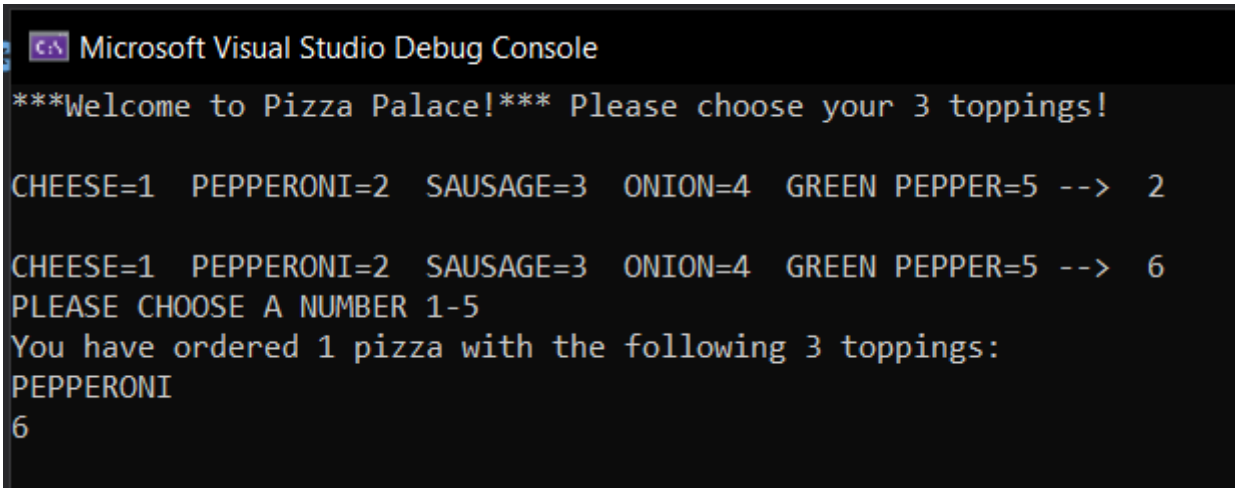
The break statement can be used to jump out of a loop! To use the break; statement, simply make some condition that needs to be met, followed by the break; and you will jump out of the loop.

Let's take the pizza ordering program we just created. Our topping choices for the user are numbered 1-5 and obviously we do not want them typing in the number 6 because that would crash our entire program! Let's add a condition to our code that says "if the user input is greater than 5 we will break; out of the loop:



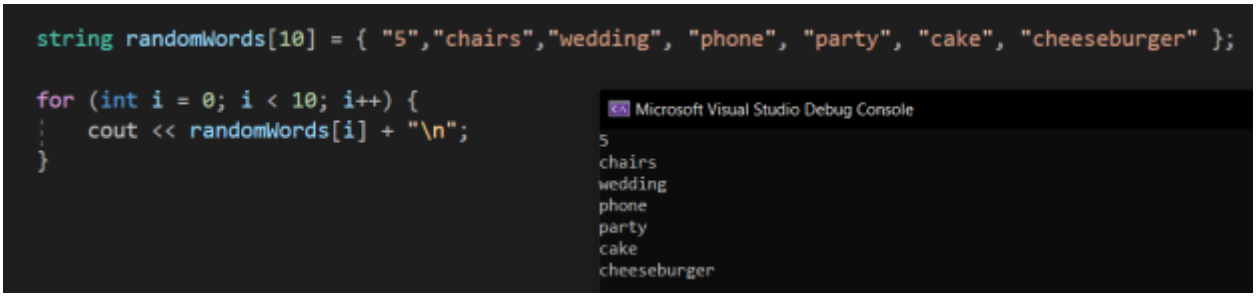
Here is what is happening. To know if the user input a number over 5 we will need to compare it (do math with it) to another number. Since our user input is always a string we must convert it to an integer. At the top of our code we will initialize/create our integer variable. Then we must convert the string to an integer. And last, we can now compare the user

input integer to see if it is greater than 5! Let's run this code and see if it breaks out of the loop when 6 is typed:

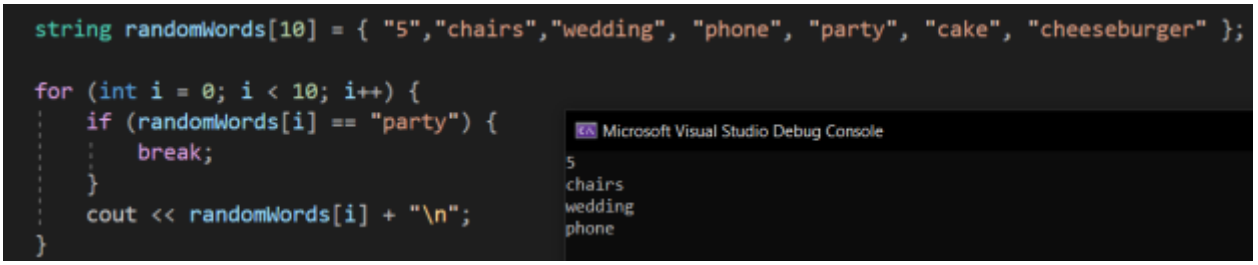

Microsoft Visual Studio Debug Console
Welcome to Pizza Palace! Please choose your 3 toppings!
CHEESE=1 PEPPERONI=2 SAUSAGE=3 ONION=4 GREEN PEPPER=5 --> 2
CHEESE=1 PEPPERONI=2 SAUSAGE=3 ONION=4 GREEN PEPPER=5 --> 6
PLEASE CHOOSE A NUMBER 1-5
You have ordered 1 pizza with the following 3 toppings:
PEPPERONI
6

Yes, our break; statement did work and did throw us out of the loop with the output statement 'please choose a number 1-5'. It ran through the rest of the code, even though it had incorrect information. This illustrates how a break; statement works, pretty simple right?!

Let's create a basic loop that will print out every word of an array, then we will break out of the loop if the string is equal to "party":

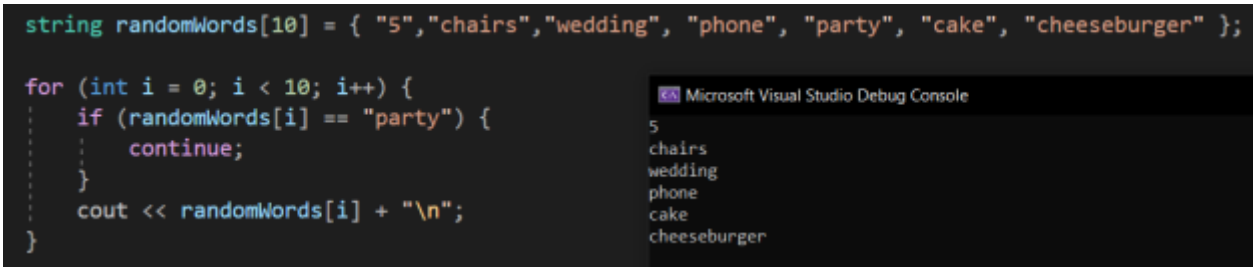


Now let's add a break statement to our code:



As soon as our if conditional statement equaled true, it broke out of the loop! Now let's learn about the continue; statement and add that to our code.

The continue; statement breaks one iteration in our loop. This means that the loop will skip the iteration, then keep moving through the loop. Let's change our previous break; statement to a continue; statement and see what happens:



So when we hit the loop where party was supposed to be printed our code said "Hey there! Don't do whatever you normally would do in this loop and move on! Just skip this and pretend like it never happened!"

FUN WITH FUNCTIONS

So far, we have worked with tons of functions. The main() is a function, the for(), while() and if()'s are all functions as well. Since these are built-in functions, there are certain parameters that you must pass to them or they will not work. For example, the for() function takes 3 specific parameters (initializer counter; conditional statement; counter increment) and it will not take anything it is not created to expect.

Did you know we can create our own functions to perform actions?! Think of it as being similar to how we create and name our variables. The way you write (declare) your own function is:

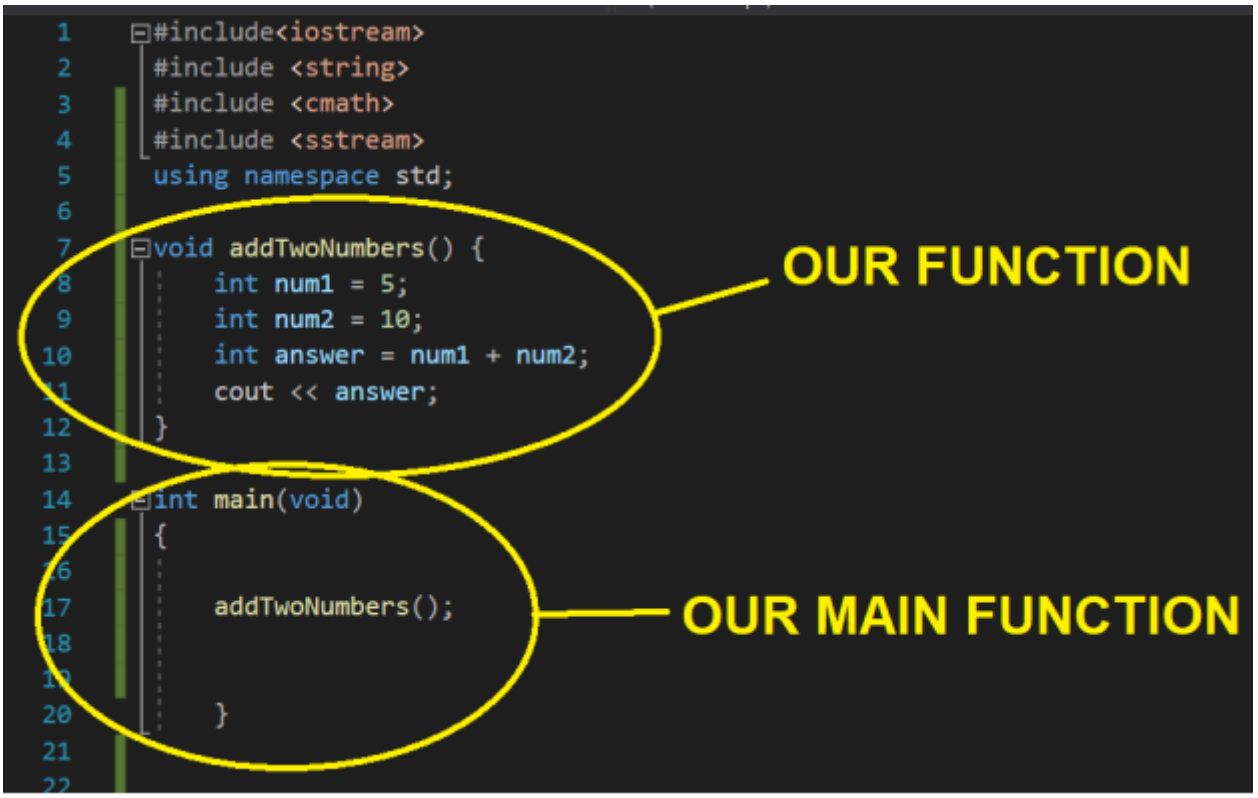
```
void myFunction( );  {
}
```

The keyword 'void' is the type of function. Void means that the code within your function will run and you are not returning/sending any information back to the main(). If we wanted to run our function and return an integer, string, boolean, etc then we would declare our function type in those ways.

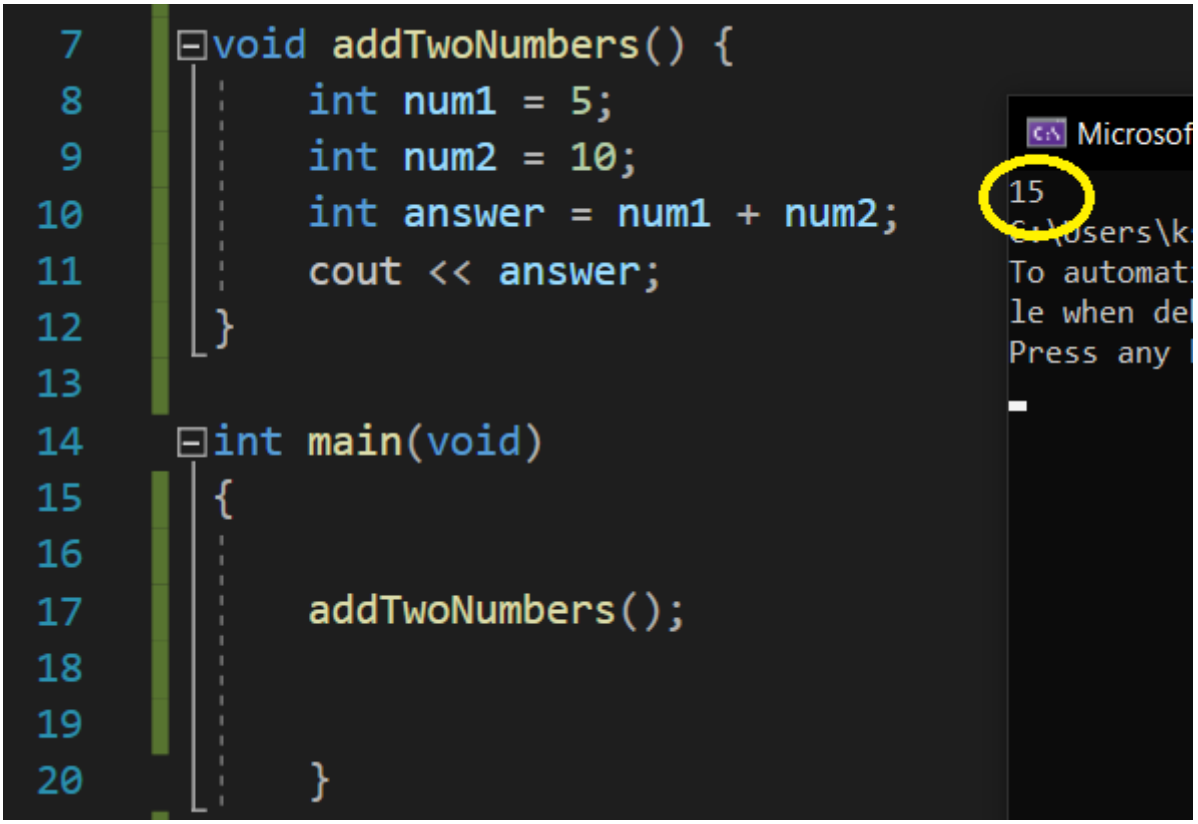
About your function name... you can name your function anything you want! Obviously you want to create a descriptive name for your function.

Within your function's parenthesis ( ) you can leave it blank or you can specify exactly what data you want. Lastly, your functions must be out of the main() function! Make sure your function is not contained within the braces { } of your main().

It is important to know that just like normal built-in functions, your functions will do nothing unless you call them. Let's create a simple function that adds 2 numbers then prints out the answer. Try it out!



As you can see, our function's world is within their opening and closing curly braces { }. Also, any functions you create need to be placed above your main() function. Our main() function is where the computer reads our program, top to bottom, so let's call our function and see if it does what it is supposed to!



So, the computer goes to our main and reads the first line, which is calling our function. The computer exits the main() and follows our addTwoNumbers() function and executes the code there.

It is important to understand that you want your main() function to almost read like a book. If someone were to come look at the program you created, they should be able to look at your main() and generally understand exactly what your program is doing.

For example, take our previous pizza ordering program. Using our own functions our main() could look like this:

getUserToppingChoices();

displayToppingChoices();

displayArrivalTime();

See how easy it is to understand what our program was doing in this way? Otherwise, even people that look at their own code

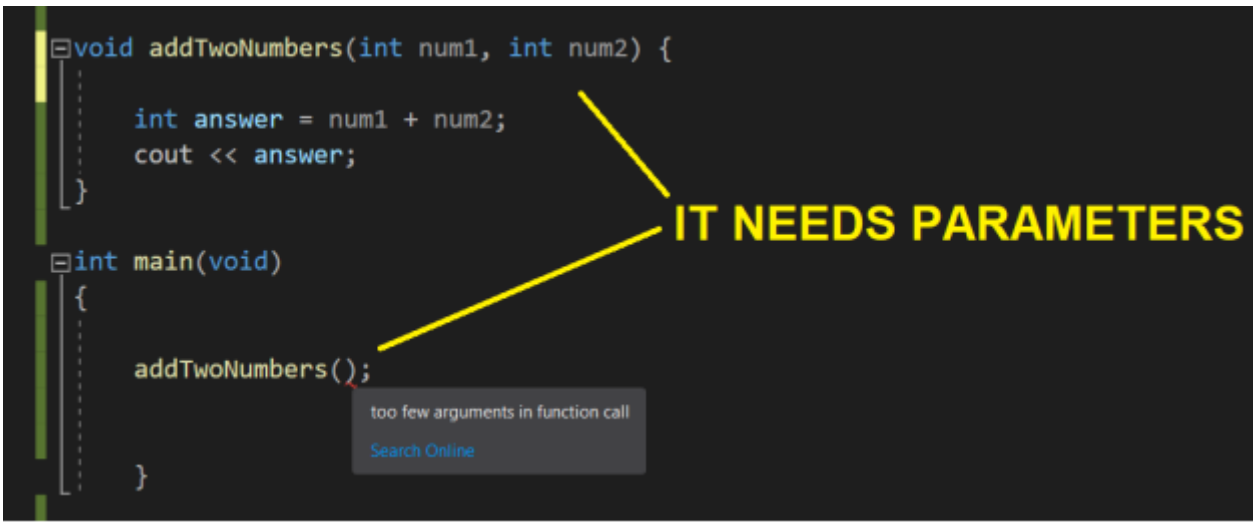
later can get overwhelmed by 5,000 lines of jumbled code in the main()!!!

So if we wanted to displayArrival() in multiple places in our program, instead of having to duplicate the code for doing that multiple times, we could instead just call the displayArrival() function. This is also advantageous because if we wanted to change the way our program displays arrival details then we could just update the code in the function and the program would automatically update the way it displays arrival details all throughout our program!

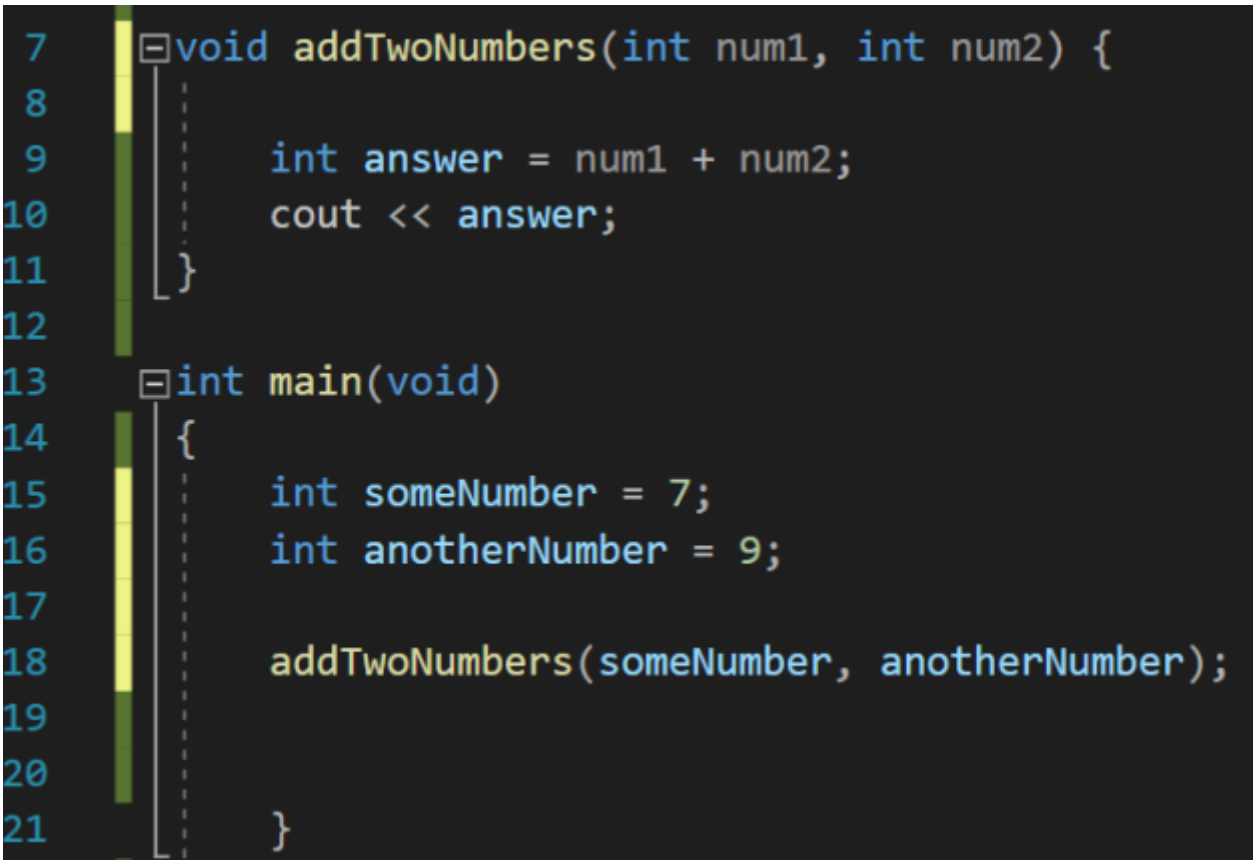
Methods are powerful not only because they allow us to re-use the same code over and over, but because they also allow us to pass them information. We can send information to our function by placing it within the parenthesis ( ) of our function! These are called parameters and all you need to do is define what type of parameters your function needs within the actual function. Let's say we want to send 2 numbers to our addTwoNumbers() function.

```
void addTwoNumbers(int num1, int num2) {

    int answer = num1 + num2;
    cout << answer;
}
```

We just said 'Hey, when you call this function I am going to need 2 integers! If you do not give me 2 integers I will not work!". Now that we just required 2 integer parameters let's take a look at the error we are getting in the main():



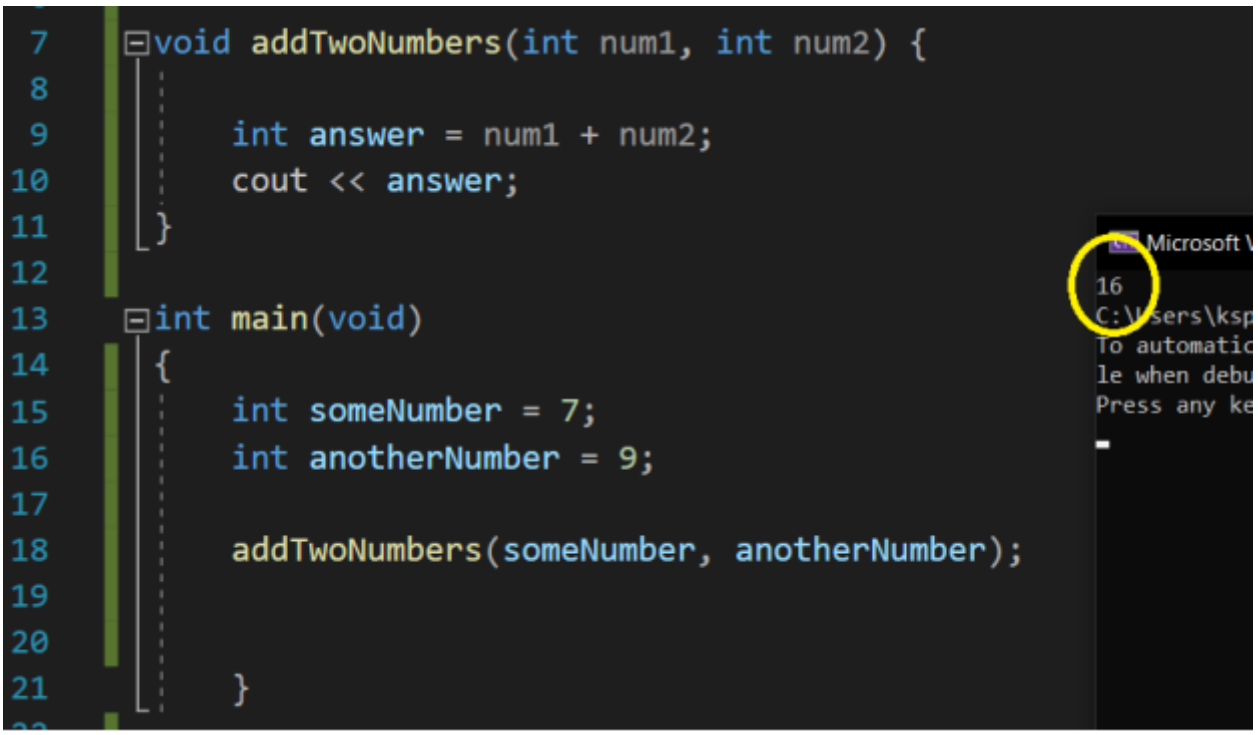
In our main(), let's declare two integer variables then place them in our function call:



Now there is no error because we have passed 2 integers into our function! You can require any amount of parameters into

your function, so for example, you could require a string, 2 integers, a boolean then another string, etc.

Here is the outcome when we run our code:



The next thing we can do with functions is pass back information to our main()!

What if we want to ask the user for the 2 numbers, pass them back to the main and assign them to our someNumber and anotherNumber variables? The first thing we would need to do is set our variables to empty:

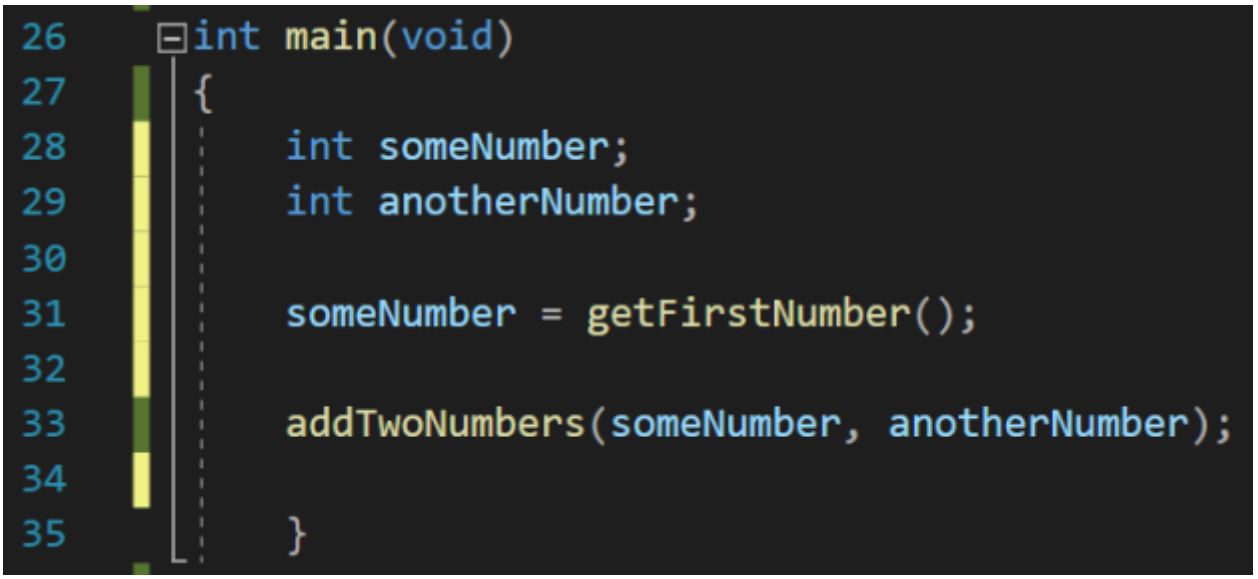
```
15  int main(void)
16  {
17      int someNumber;
18      int anotherNumber;
19
20      addTwoNumbers(someNumber, anotherNumber);
21
22
23      }
```

When we create a function and want to return a value, we must replace the word void with the type of value we will be returning to the main(), then add a return statement at the end of our function. Let's create our first function and ask our user for the first number.

```
13  int getFirstNumber() {
14      string userInput;
15      int userNumberChoice;
16
17      cout << "Please type a number: ";
18      getline(cin, userInput);
19
20      stringstream intValue(userInput);
21      intValue >> userNumberChoice;
22
23      return userNumberChoice;
24  }
```

Here you can see that we declared our function as an integer type, meaning that our function will be returning an integer value to the main(). Next, our code is what we have previously done by getting the user string input and converting that string to an integer. Lastly, our return statement will return the value of our integer variable userNumberChoice!

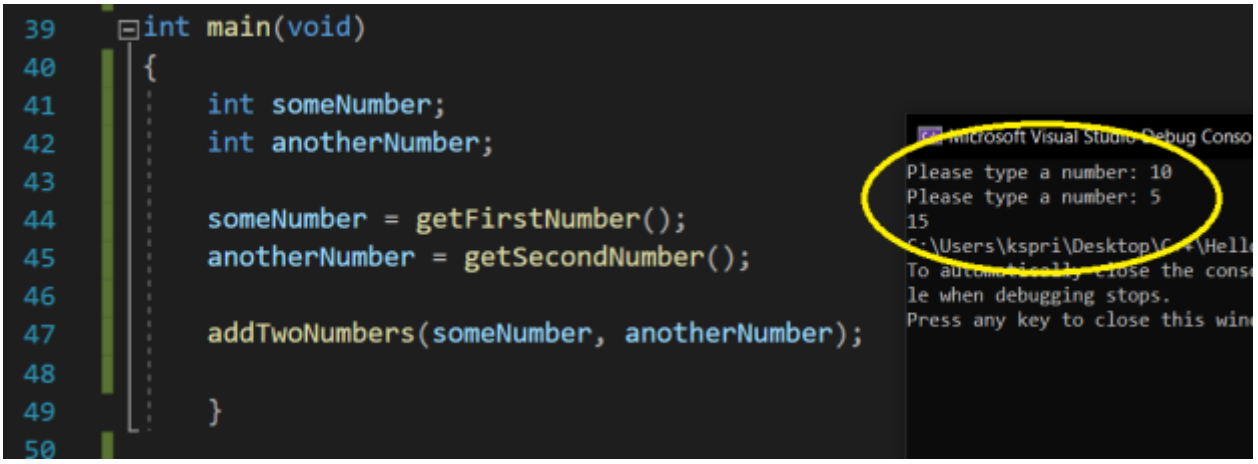
So what do we do with this returned value? We need to assign this returned value to our variable someNumber of course! Here is how we can do this:



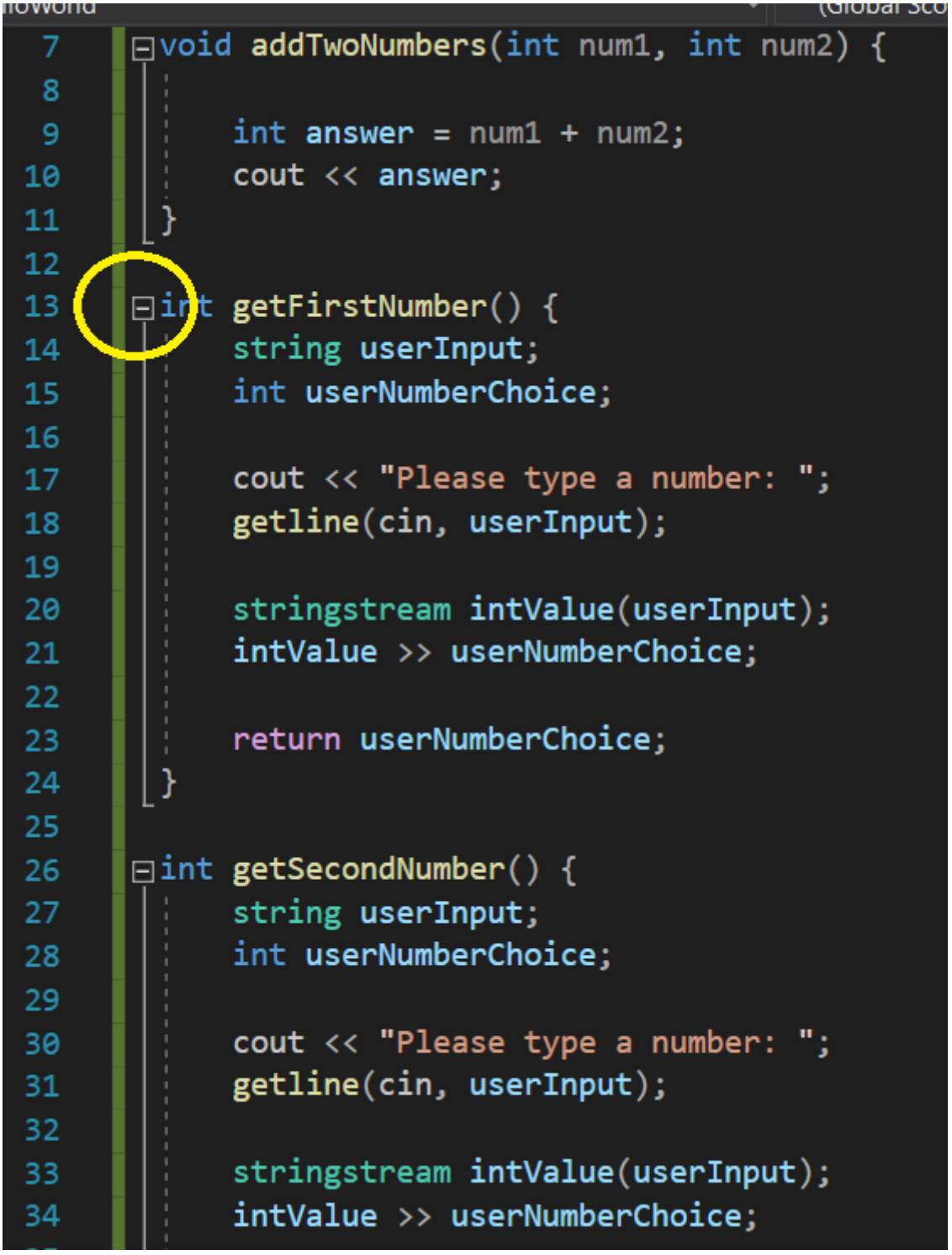
Since we know our getFirstNumber() function holds a return value, we simply assign that function to our integer variable. Now let's do this with a second method and assign the return value to anotherNumber variable:

```
26  int getSecondNumber() {
27      string userInput;
28      int userNumberChoice;
29
30      cout << "Please type a number: ";
31      getline(cin, userInput);
32
33      stringstream intValue(userInput);
34      intValue >> userNumberChoice;
35
36      return userNumberChoice;
37  }
38
39  int main(void)
40  {
41      int someNumber;
42      int anotherNumber;
43
44      someNumber = getFirstNumber();
45      anotherNumber = getSecondNumber();
46
47      addTwoNumbers(someNumber, anotherNumber);
48
49      }
```

Now let's run your code and check out the result:



Now that we have multiple functions, let me show you how to collapse them so you are not forced to look at tons of functions before seeing your main:



```
7   void addTwoNumbers(int num1, int num2) {
8
9       int answer = num1 + num2;
10      cout << answer;
11  }
12
13  int getFirstNumber() {
14      string userInput;
15      int userNumberChoice;
16
17      cout << "Please type a number: ";
18      getline(cin, userInput);
19
20      stringstream intValue(userInput);
21      intValue >> userNumberChoice;
22
23      return userNumberChoice;
24  }
25
26  int getSecondNumber() {
27      string userInput;
28      int userNumberChoice;
29
30      cout << "Please type a number: ";
31      getline(cin, userInput);
32
33      stringstream intValue(userInput);
34      intValue >> userNumberChoice;
```

You can click on the (-) collapse button to hide all of the code for your functions to end up with a result similar to this:

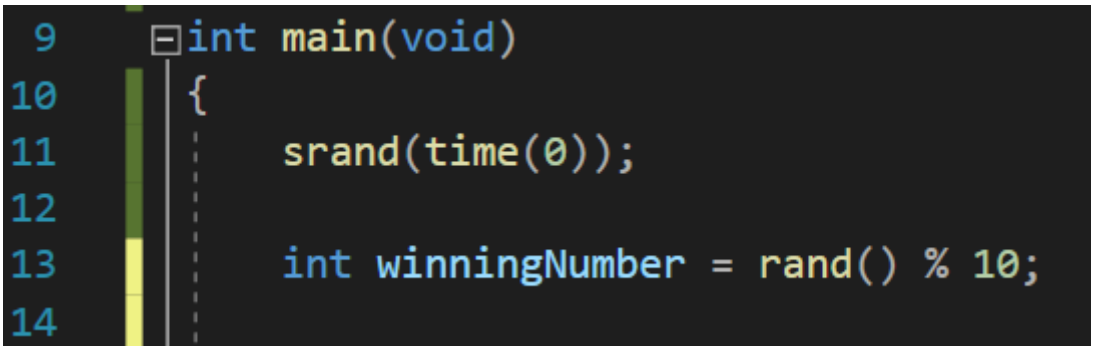
```
 7   void addTwoNumbers(int num1, int num2) { ... }
12
13   int getFirstNumber() { ... }
25
26   int getSecondNumber() { ... }
38
39   int main(void)
40   {
41       int someNumber;
42       int anotherNumber;
43
44       someNumber = getFirstNumber();
45       anotherNumber = getSecondNumber();
46
47       addTwoNumbers(someNumber, anotherNumber);
48
49       }
```

Now that you understand the fundamentals of functions, let's recreate our Guess The Number game, but first, let's import and use a pre-built code to randomly generate our winning number. To start, add #include to your header:

```
1   #include<iostream>
2   #include <string>
3   #include <cmath>
4   #include <sstream>
5   #include <cstdlib>
6   using namespace std;
```

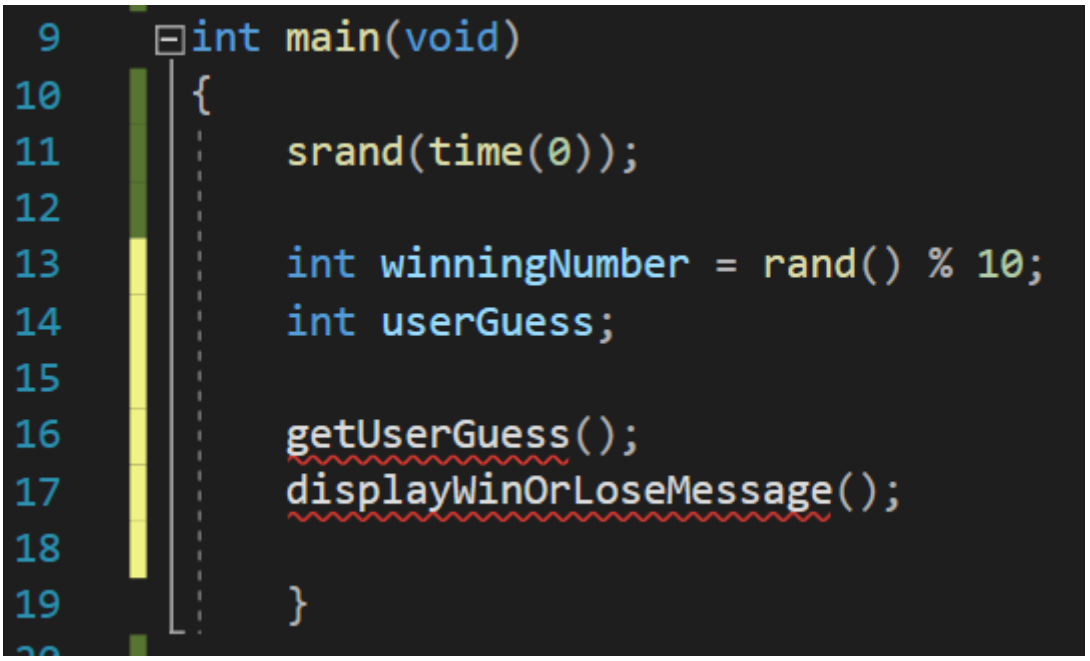
This will allow us to use the srand() function to initialize the rand() function. The rand() function will produce a random number. This random number can be anything but we want a random number between one and ten. You can make C++ generate random numbers from a unique range by using the %

modulus operator. This operator lets you set the maximum value that can be generated. The following example shows how you can return a random number between 0 and 10:



If you wanted a number between 0 and 57 then you would state: rand() % 57 and so on.

Let's make a brief outline of our functions in the main(), then we will create the functions:



Clearly we are getting errors because we have not created our functions yet. Just looking at our outline we know that we will need to return a value from our getUserGuess() function and assign it to our userGuess variable. We can also see that we will need to pass in our winningNumber and userGuess variables to our displayWinOrLoseMessage() function. Try it yourself before moving on!

Here is the code in our main():

```
31  int main(void)
32  {
33      srand(time(0));
34
35      int winningNumber = rand() % 10;
36      int userGuess = getUserGuess();
37      displayWinOrLoseMessage(winningNumber, userGuess);
38
39      }
```

Now let's take a look at the 2 functions:

```
 8  int getUserGuess()
 9  {
10      string userInput;
11      int userNum;
12
13      cout << "*****GUESS THE NUMBER GAME*****\n";
14      cout << "Type a number between 0 and 10: ";
15      getline(cin, userInput);
16      stringstream intValue(userInput);
17      intValue >> userNum;
18      return userNum;
19
20  }
```

```
22  void displayWinOrLoseMessage(int correctNum, int userNumGuess) {
23      if (correctNum == userNumGuess) {
24          cout << "CONGRATULATIONS!  YOUR GUESS OF " << userNumGuess << " MATCHES THE WINNING NUMBER " << correctNum;
25      }
26      else {
27          cout << "SORRY!  YOUR GUESS OF " << userNumGuess << " DID NOT MATCH THE WINNING NUMBER " << correctNum;
28      }
29  }
```

And here is the output:

```
Microsoft Visual Studio Debug Console
*****GUESS THE NUMBER GAME*****
Type a number between 0 and 10: 8
SORRY!  YOUR GUESS OF 8 DID NOT MATCH THE WINNING NUMBER 6
```

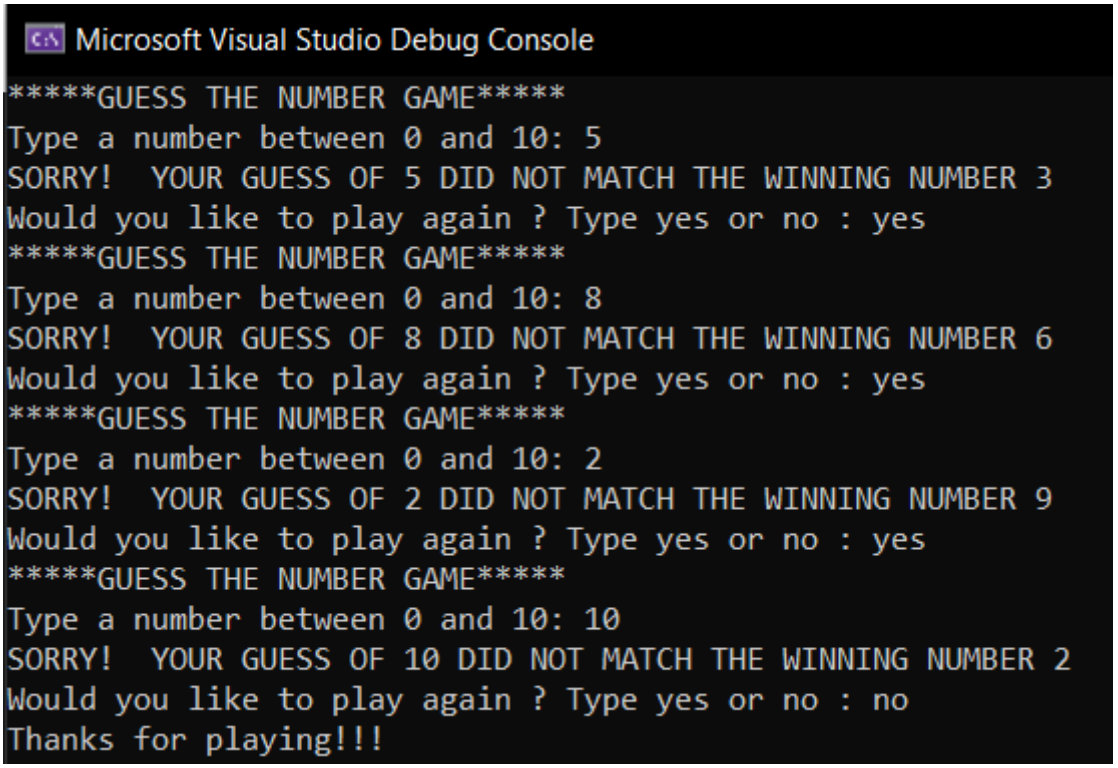
That was fun! Now what if we wanted to ask the user if they would like to play again? Let's make a function that asks the user this question and stores that answer in a variable:

```
29      }
30
31      string playAgain() {
32          string answer;
33          cout<< "Would you like to play again ? Type Yes or No : ";
34          getline(cin, answer);
35          return answer;
36      }
37      int main(void)
38      {
39          srand(time(0));
40
41          int winningNumber = rand() % 10;
42          int userGuess = getUserGuess();
43          displayWinOrLoseMessage(winningNumber, userGuess);
44          string userPlayAgain = playAgain();
45
46          }
47
```

Now, using logic, we should loop our game if their answer is yes and stop the game if their answer is no. If you are wondering which loop to use just think of your goal... as long as the userPlayAgain variable is "yes" you want to loop:

```
37  int main(void)
38  {
39      srand(time(0));
40
41      int winningNumber = rand() % 10;
42      int userGuess = getUserGuess();
43      displayWinOrLoseMessage(winningNumber, userGuess);
44      string userPlayAgain = playAgain();
45
46      while (userPlayAgain == "yes") {
47          srand(time(0));
48
49          int winningNumber = rand() % 10;
50          int userGuess = getUserGuess();
51          displayWinOrLoseMessage(winningNumber, userGuess);
52          userPlayAgain = playAgain();
53      }
54
55          cout << "Thanks for playing!!!";
56
57      }
```

So, while userPlayAgain is equal to yes our game will keep looping. As soon as that variable changes it will immediately exit the loop and print out “Thanks for playing!!!” - Here is an output:


Microsoft Visual Studio Debug Console
*****GUESS THE NUMBER GAME*****
Type a number between 0 and 10: 5
SORRY! YOUR GUESS OF 5 DID NOT MATCH THE WINNING NUMBER 3
Would you like to play again ? Type yes or no : yes
*****GUESS THE NUMBER GAME*****
Type a number between 0 and 10: 8
SORRY! YOUR GUESS OF 8 DID NOT MATCH THE WINNING NUMBER 6
Would you like to play again ? Type yes or no : yes
*****GUESS THE NUMBER GAME*****
Type a number between 0 and 10: 2
SORRY! YOUR GUESS OF 2 DID NOT MATCH THE WINNING NUMBER 9
Would you like to play again ? Type yes or no : yes
*****GUESS THE NUMBER GAME*****
Type a number between 0 and 10: 10
SORRY! YOUR GUESS OF 10 DID NOT MATCH THE WINNING NUMBER 2
Would you like to play again ? Type yes or no : no
Thanks for playing!!!

Now we're talking! You are now realizing how to make the "bones" of a program! This is obviously important and you must understand programming logic before adding all the pretty graphics and user interface components!

OOP CLASSES

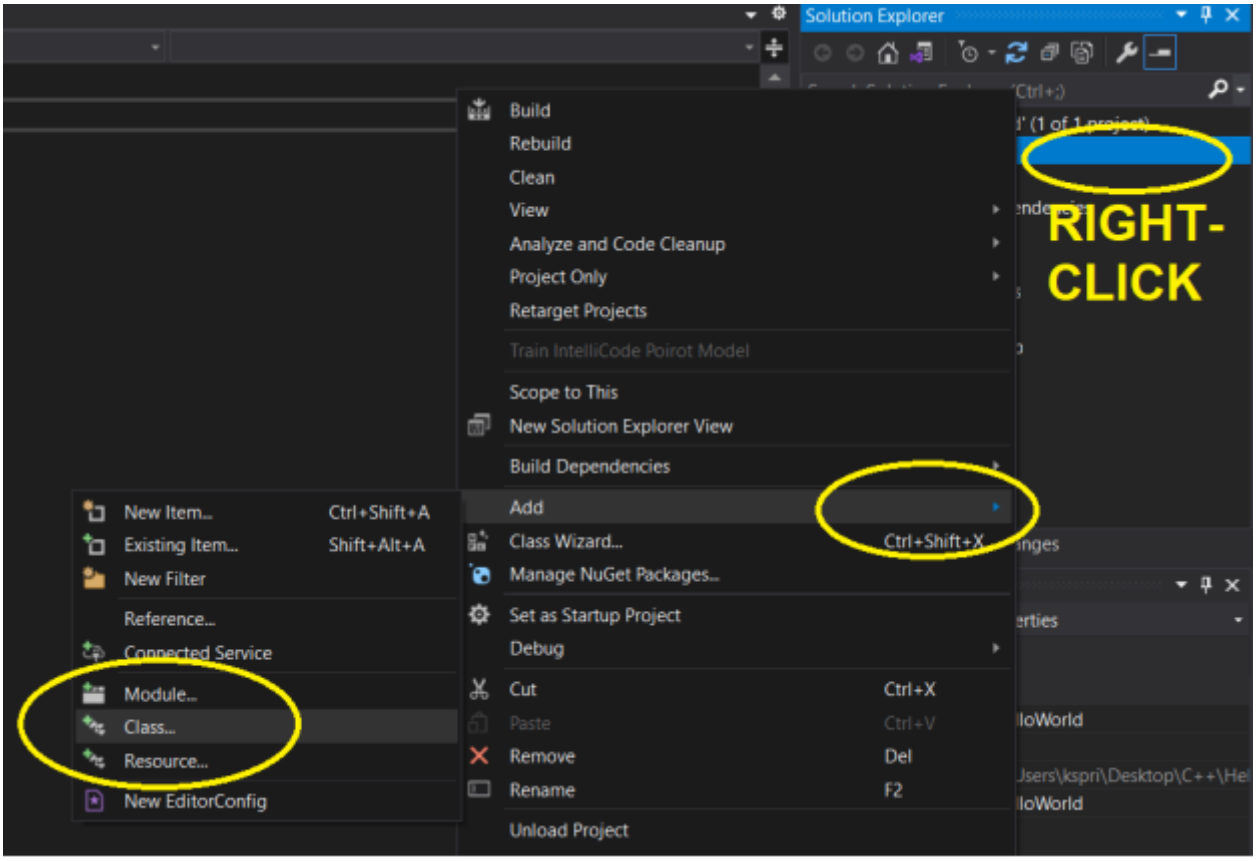
Object Oriented Programming

Class: A class in C++ is the building block that leads to Object-Oriented programming (OOP). Aclass is a user-defined data type, which holds its own variables and functions. These variables and functions can be accessed and used by creating an instance of that class.

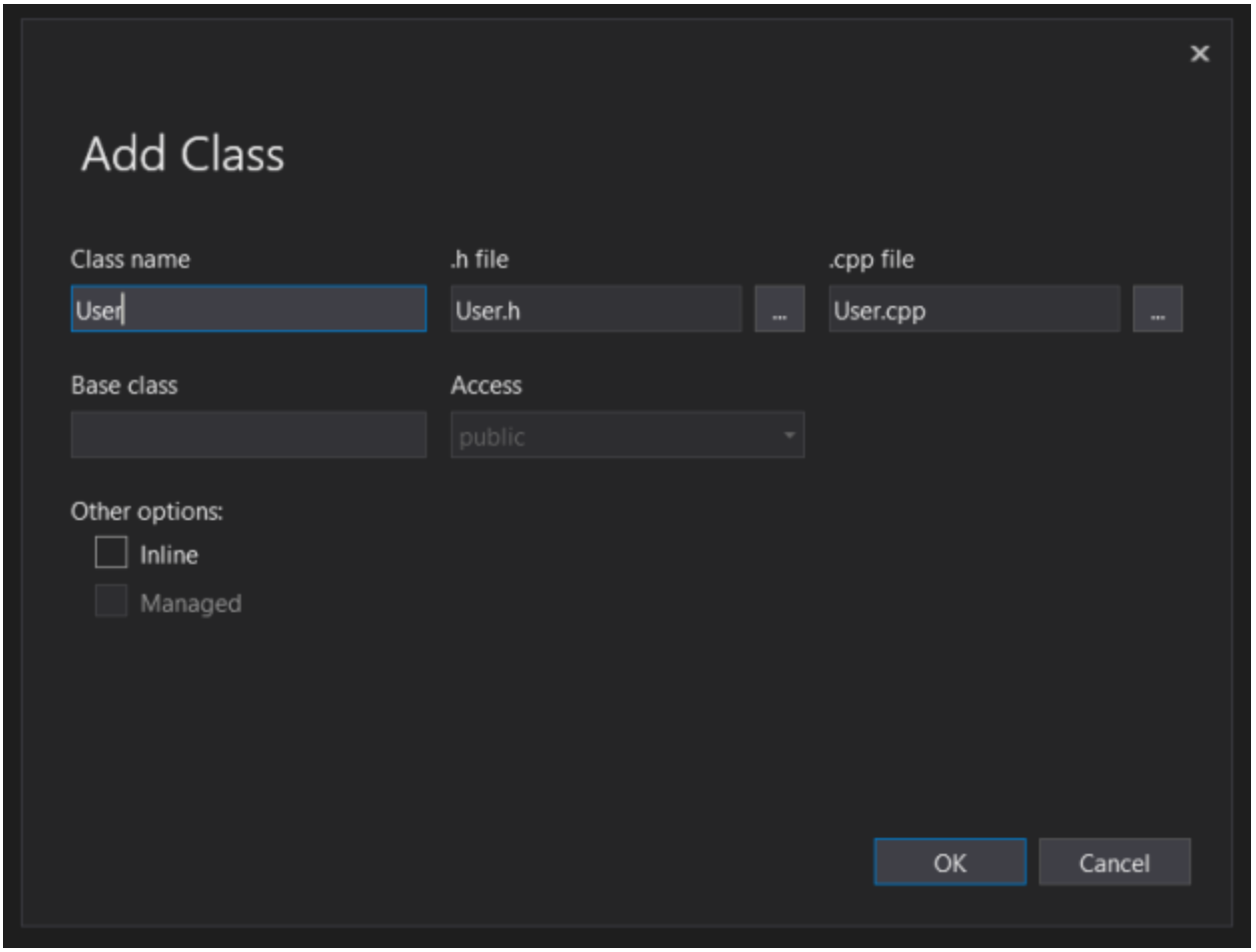
For example, say we create a Class of Cars. There may be many cars with different names, brands, etc, but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, color etc. So here, Car is the class and wheels, speed limits and color are the Car Class properties.

When we create an instance of a class it is called an Object. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated, making your program faster.

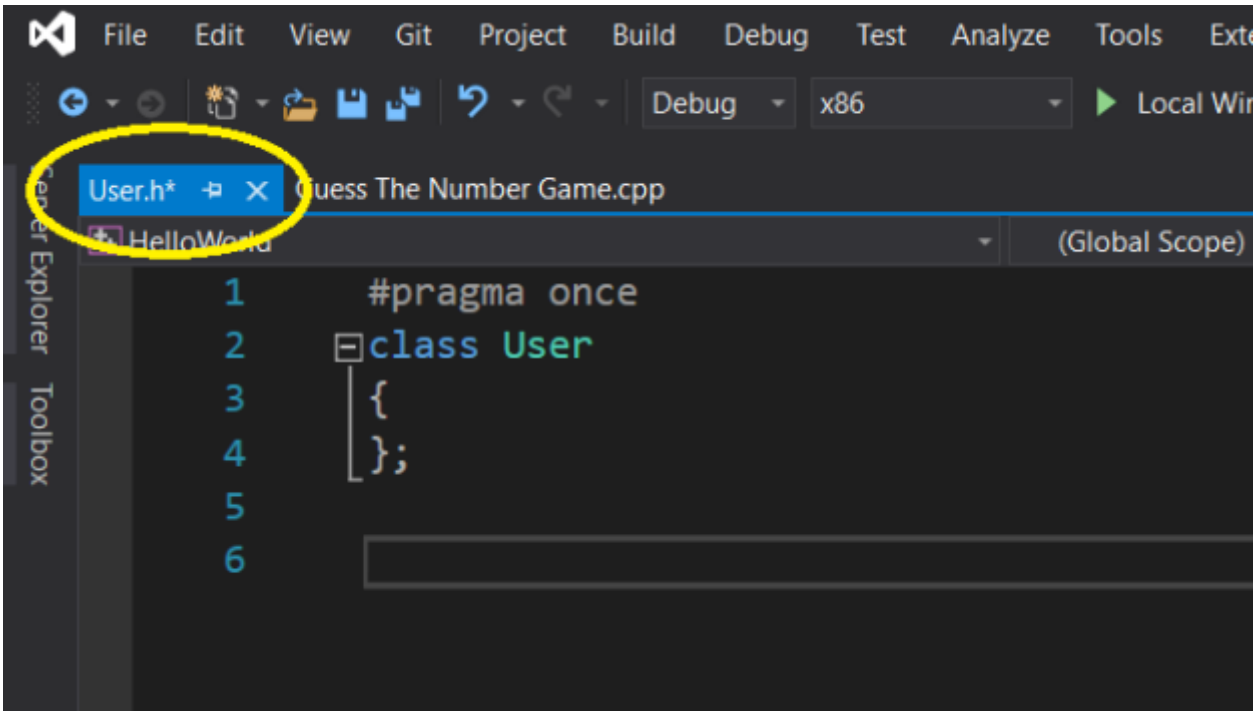
Let's get into it and create a class! First, right click in your solution explorer title then choose add new class:



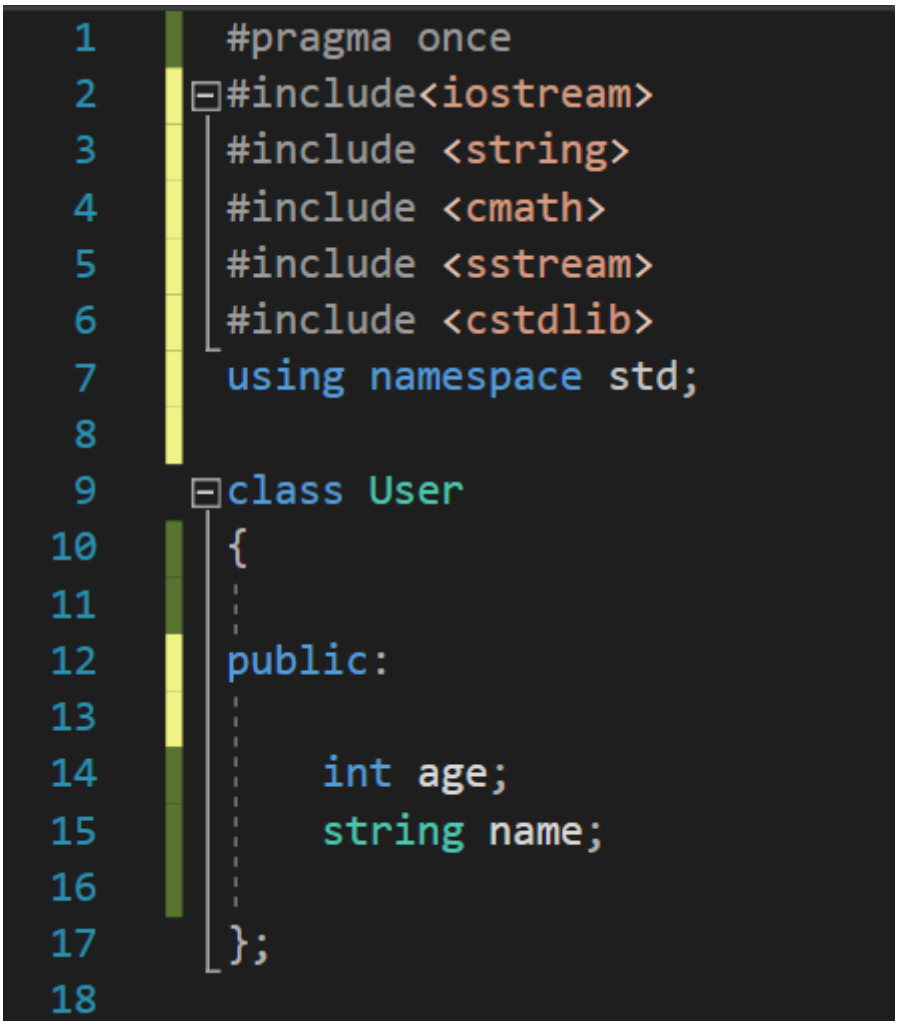
You can name your class anything - for our example I will name it User:



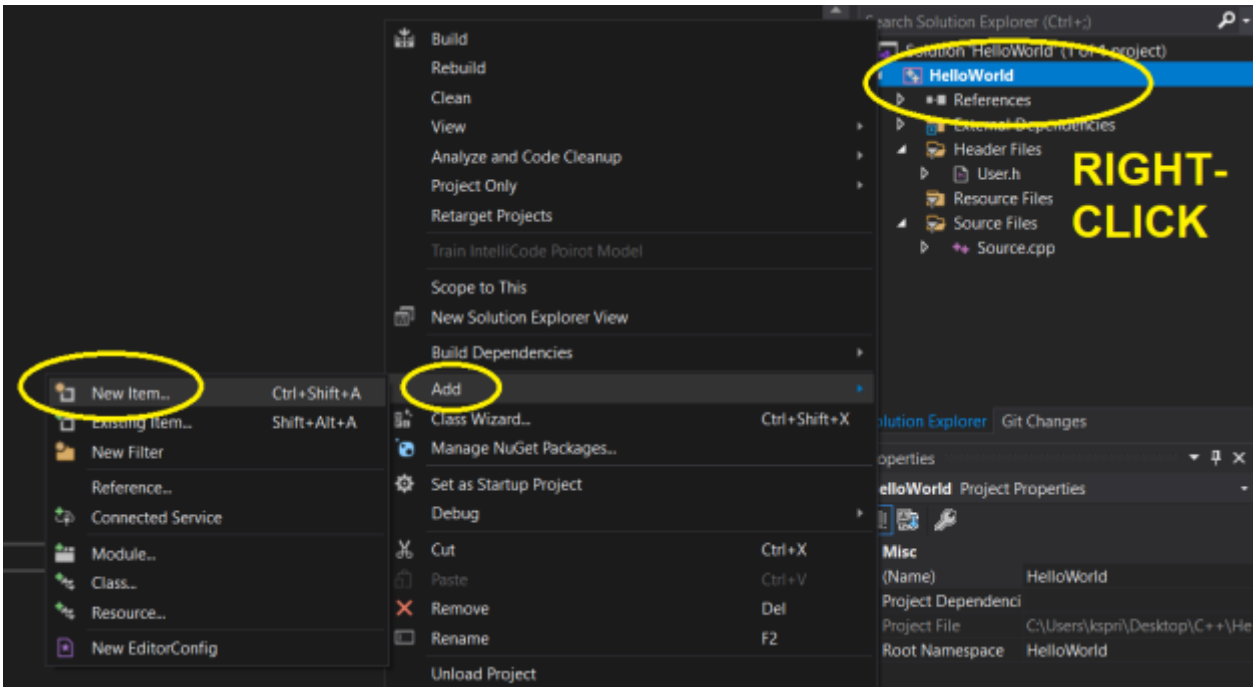
Your class will show up in it's own tab within your IDE:

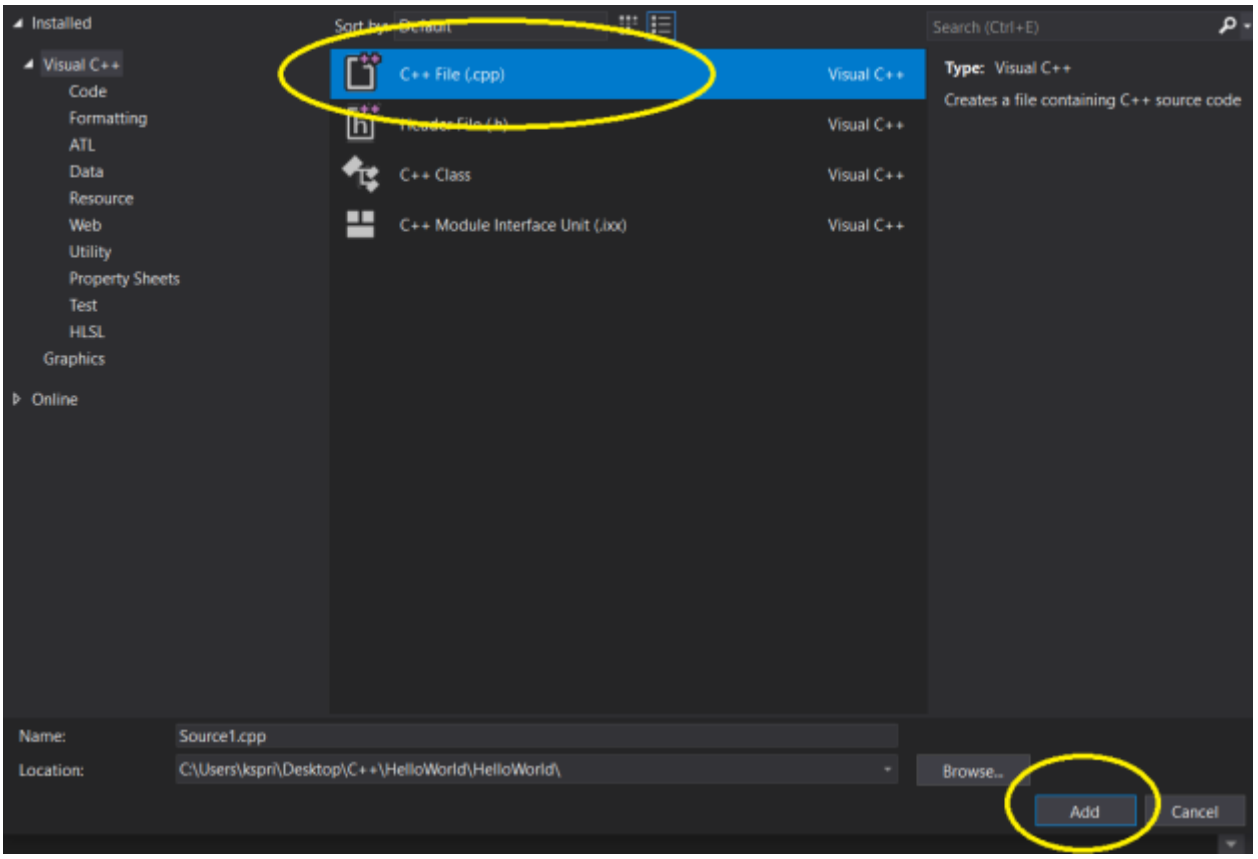


The class keyword is used to create a class called User. Within the class we will state our access specifier keyword, which specifies if the variables (attributes) and functions within this class are accessible from outside the class. Let's give our class a public access specifier and make some attributes that all of our users will have, like name and age:

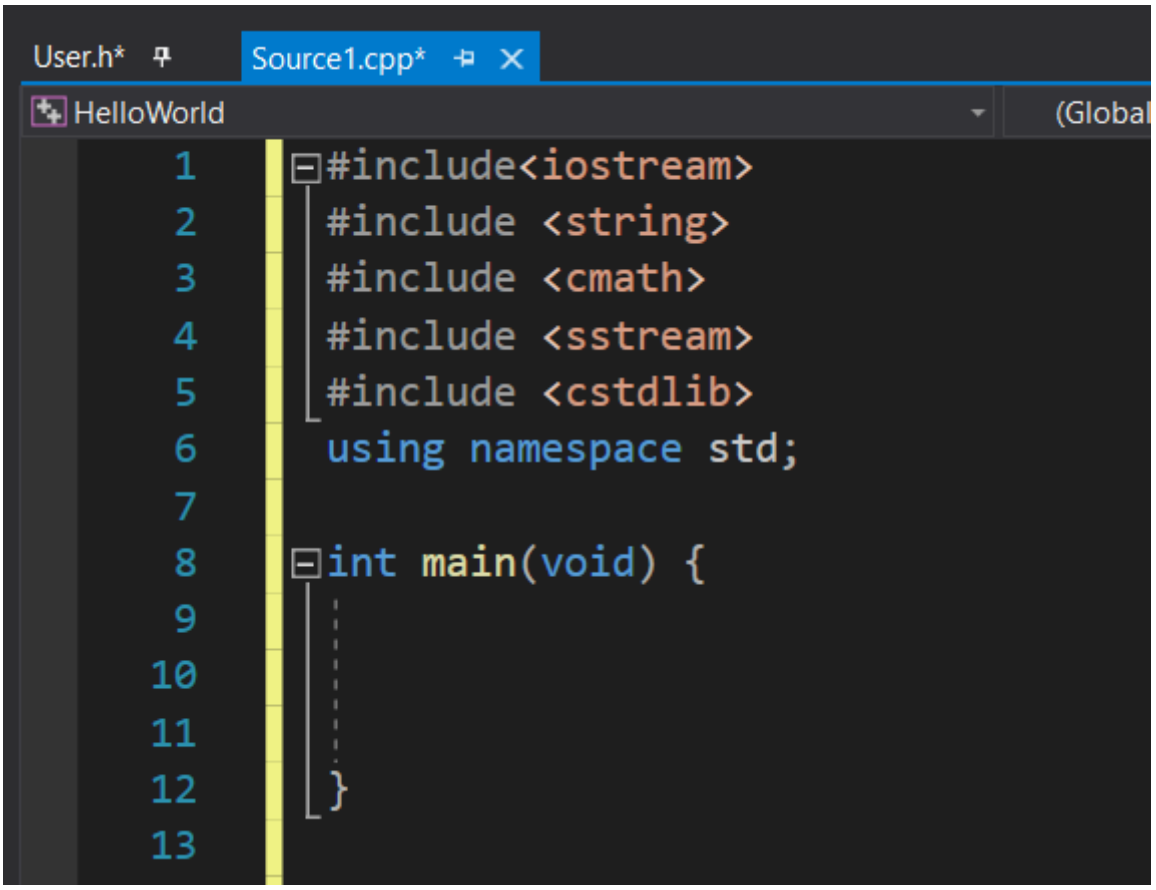


You could also set this to private: if you do not want anything outside your class viewing it. Now, let's create a new source .cpp file so we can start playing with our class. Right-click on your main solution/program name and click add, new, then choose .cpp file. Alternatively you can just create a completely new project and start from there.

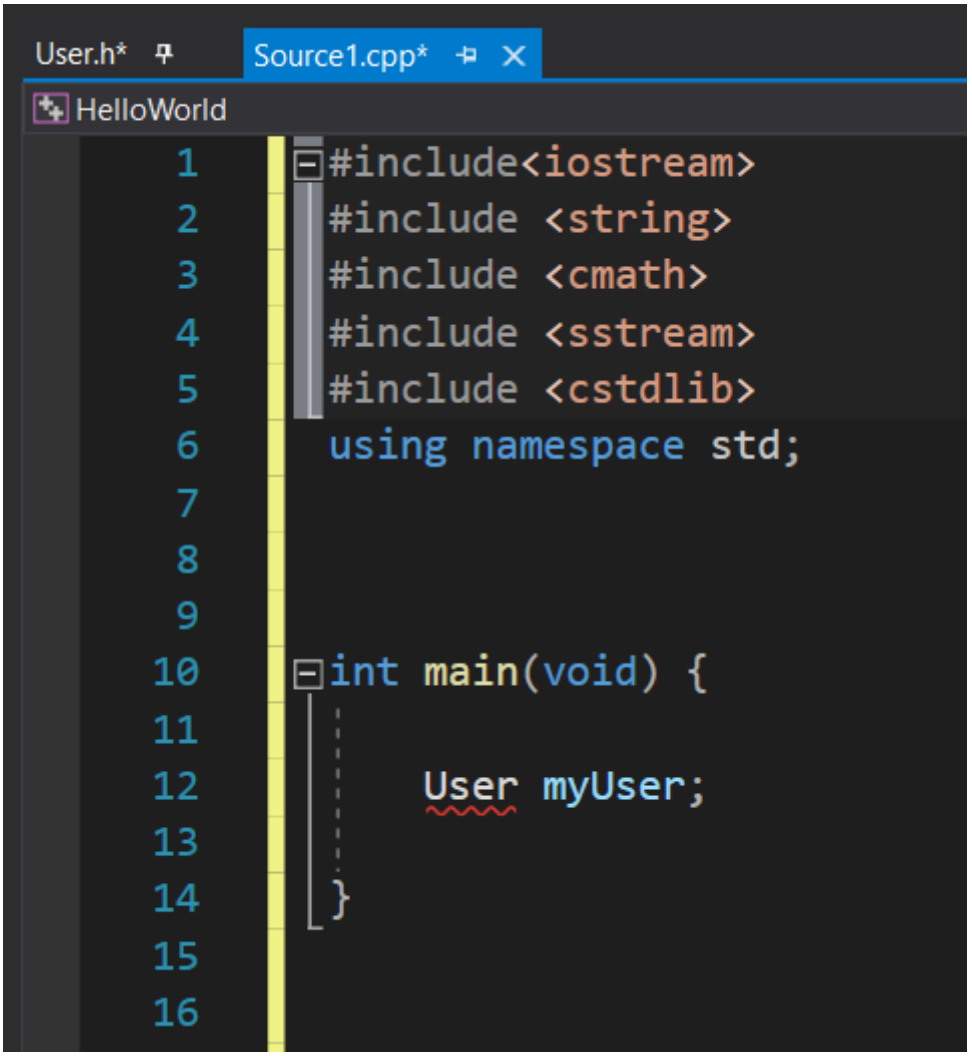




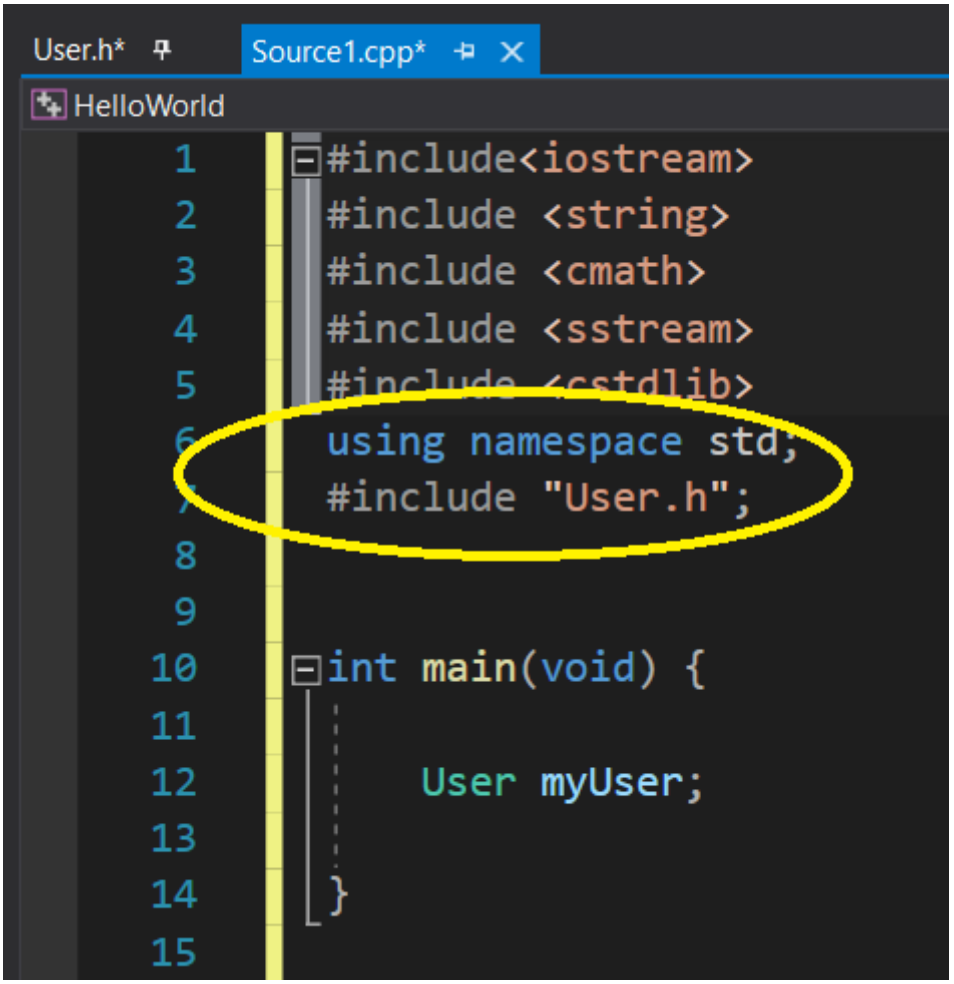
This will create an empty source file, so let's #include all of our header statements in case we need them. Then let's make a main():



Now, let's instantiate/create an object from our user class. To instantiate an object you simply write the class name, then name your object whatever you want:



Uh-oh! Why are we getting an error?! We need to include our class file name in our main() header, that's why! To do this you will type #include followed by the actual file name in quotes, in our case it is "User.h"



There, you have now properly instantiated/created an object from your very own class! Now we will set some attributes to our new myUser object. Remember, in our class we made 2 attributes, one for age and one for name. To do this, you write your class name followed by the dot (.) operator, followed by an attribute that we already defined within our class, then assign a value to it, just like we do with normal variables:

```
10  int main(void) {
11
12      User myUser;
13
14      myUser.age = 38;
15      myUser.name = "Shawn";
16
17
18
19  }
20
```

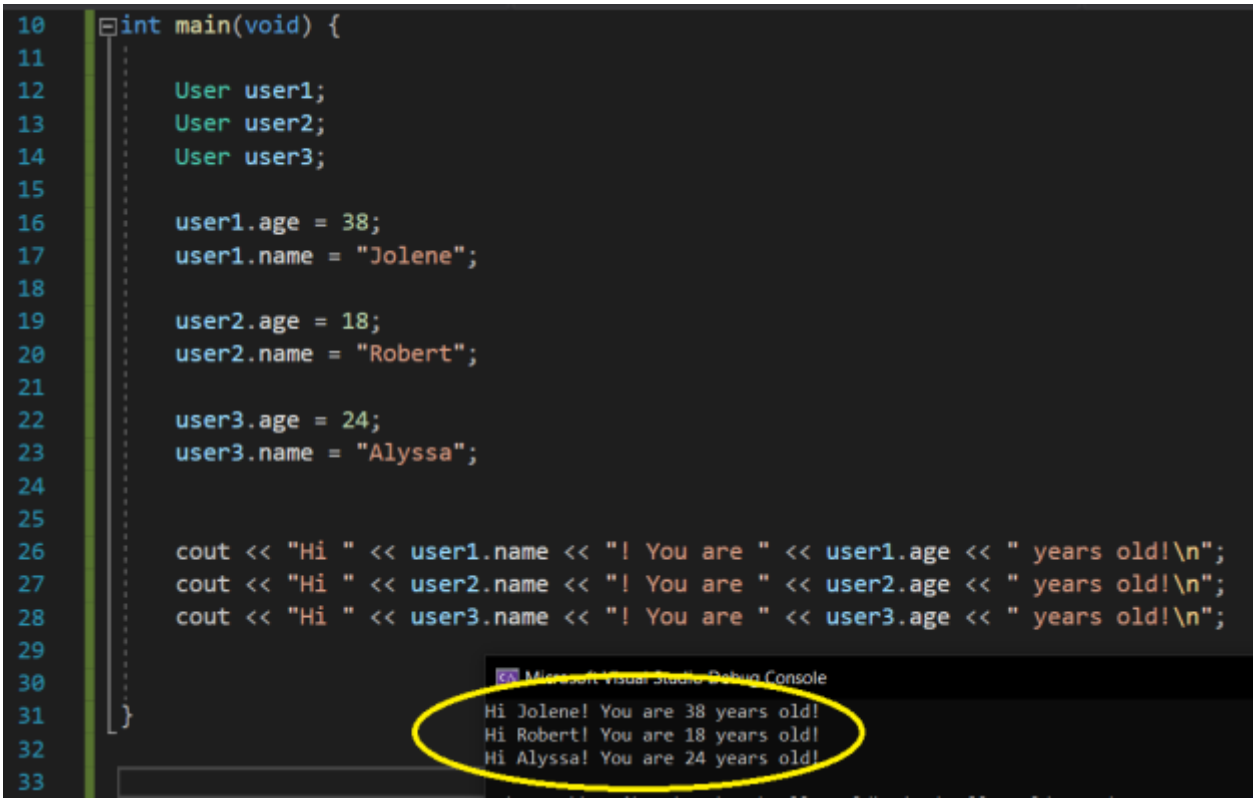
Now let's print these out:

```
10  int main(void) {
11
12      User myUser;
13
14      myUser.age = 38;
15      myUser.name = "Shawn";
16
17      cout << "Hi " << myUser.name << "!  You are " << myUser.age << " years old!";
18
19
20  }
21
```

```
Microsoft Visual Studio Debug Console
Hi Shawn!  You are 38 years old!
C:\Users\kspri\Desktop\C++\HelloWorld\Debug\HelloWorld.exe (process 16972) ex
To automatically close the console when debugging stops, enable Tools->Option
```

If you get an error when running your code then please make sure you only have one source .cpp file with a main() in it. You may have a previous file you were working on, so just right click and delete that other file. Remember, a program can only have one main().

So we instantiated our object, gave the object attributes and even called on the object attributes to display them! Next, let's try instantiating 3 users and giving them all attributes:


int main(void) {
User user1;
User user2;
User user3;
user1.age = 38;
user1.name = "Jolene";
user2.age = 18;
user2.name = "Robert";
user3.age = 24;
user3.name = "Alyssa";
cout << "Hi " << user1.name << "! You are " << user1.age << " years old!\n";
cout << "Hi " << user2.name << "! You are " << user2.age << " years old!\n";
cout << "Hi " << user3.name << "! You are " << user3.age << " years old!\n";
Hi Jolene! You are 38 years old!
Hi Robert! You are 18 years old!
Hi Alyssa! You are 24 years old!

So we have attributes for our user's name and age, but let's add some more attributes like email, username and password:

```
class User
{

public:

    int age;
    string name;
    string email;
    string username;
    string password;


};
```

Great! Now let's go back to our main() and ask our user for this input, then set our object attributes with their answers.

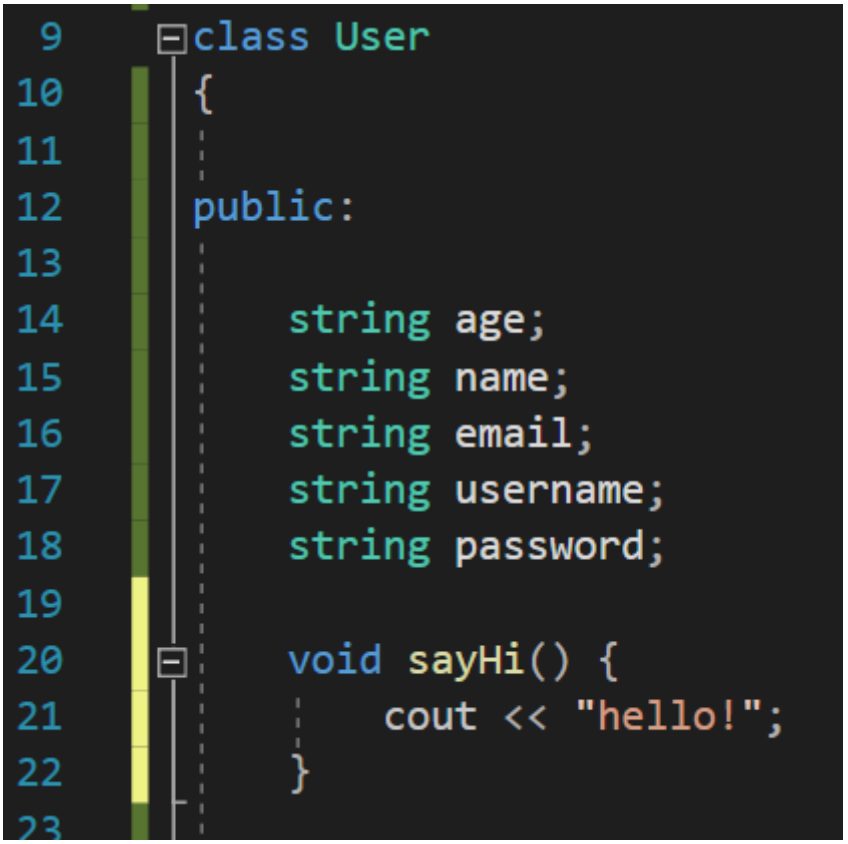
```
int main(void) {

    User user1;

    user1.name;
    user1.age;
    user1.email;
    user1.username;
    user1.password;

    cout<<"What is your name? ";
    getline(cin, user1.name);

    cout << "What is your age? ";
    getline(cin, user1.age);

    cout << "What is your email? ";
    getline(cin, user1.email);

    cout << "What would you like your username to be? ";
    getline(cin, user1.username);

    cout << "Please set your password: ";
    getline(cin, user1.password);

    cout << "Hi " << user1.name << "! You are " << user1.age << " years old!\n";
    cout << "Your email is: " << user1.email << "\n";
    cout << "Your username is: " << user1.username << "\n";
    cout << "Your password is: " << user1.password << "\n";
```

Here is the output of this code:

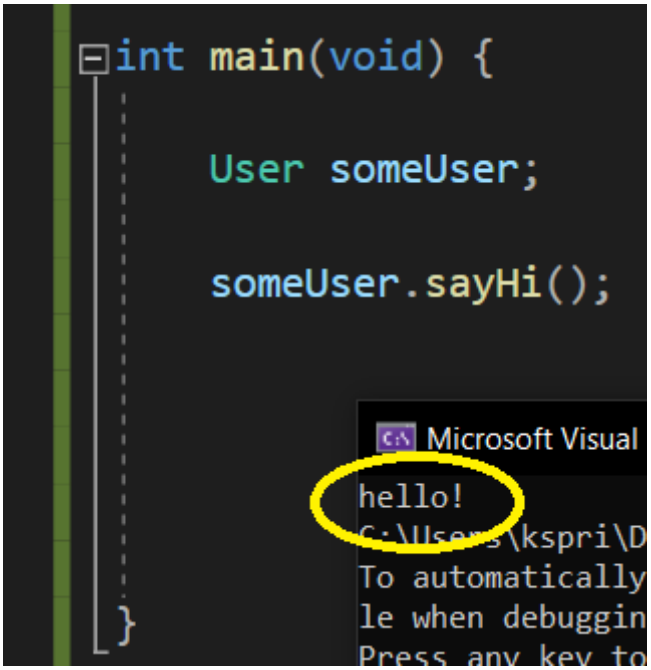
```
Microsoft Visual Studio Debug Console
What is your name? Sarah
What is your age? 34
What is your email? sarahbeth@gmail.com
What would you like your username to be? sbeth
Please set your password: lovely
Hi Sarah! You are 34 years old!
Your email is: sarahbeth@gmail.com
Your username is: sbeth
Your password is: lovely
```

CLASS METHODS

Class methods are simply functions that you put in your class. You can call on these methods the exact same way you just called on the attributes for your object(s). Let's make a method() in our class that prints out a message:

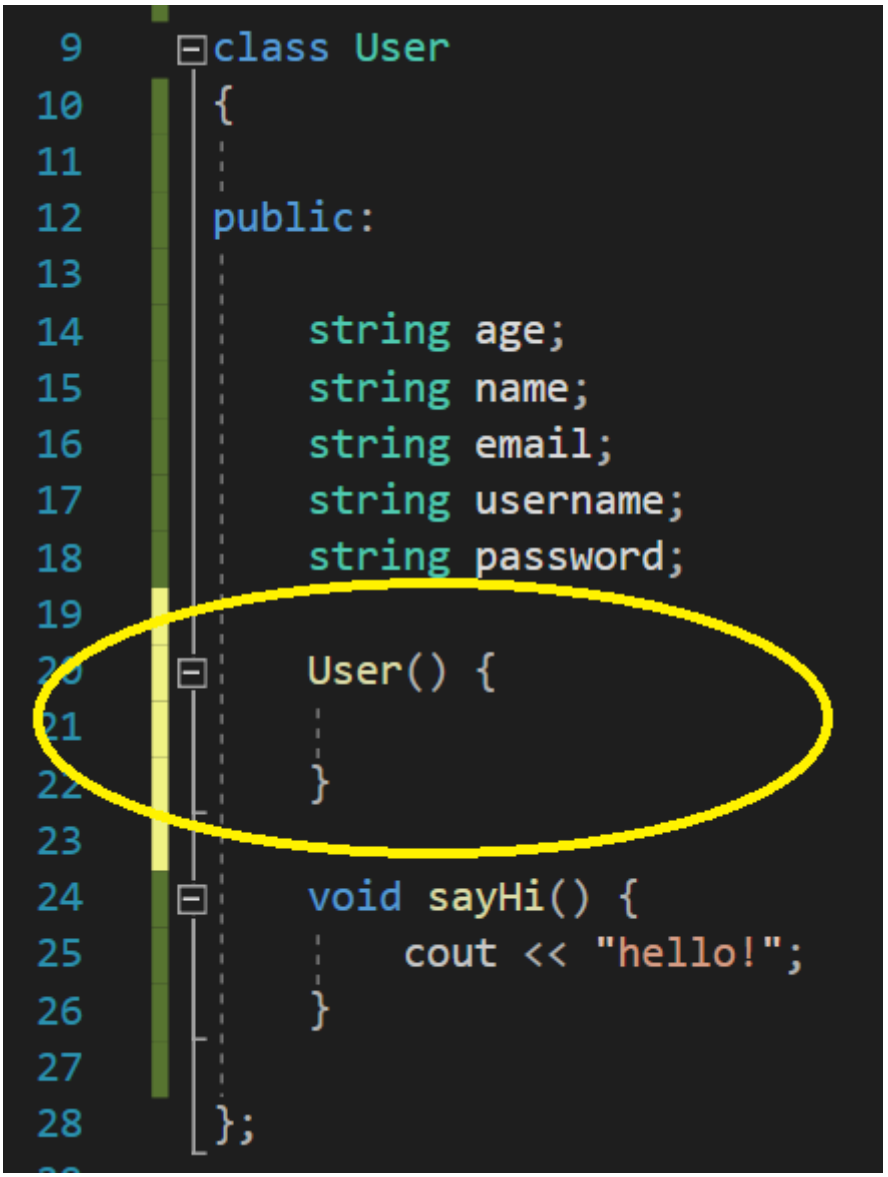


Now if we go back to the main and instantiate a new object then we can call on this class method by writing the object followed by the dot (.) operator, then the method name:


int main(void) {
User someUser;
someUser.sayHi();
}
Microsoft Visual
hello!
To automatically
le when debuggin
Press any key to

CONSTRUCTORS

A class constructor is written just like our class method but the name is always the exact same as the name of your class.



A constructor executes every single time you instantiate an object over in the main. This can be useful if you want to set specific attributes each time a new object is created. This constructor can receive parameters when you create that object.

Let's automatically set the user's username and password with something generic everytime a new user is created:

```
User() {
    username = "generic";
    password = "1234";
}
```

Now, let's create an object over in the main then print out the object's username and password to see if this works:

```
int main(void) {

    User someUser;

    cout << "Username: " << someUser.username << "\n";
    cout << "Password: " << someUser.password << "\n";
```

```
Microsoft Visual Studio Debug Console
Username: generic
Password: 1234
```

As you can see, the moment we created/instantiated an object the class constructor User() executed. This is the only thing in a class that will automatically run if you choose to place a constructor in your class.

Our class constructor can also receive parameters, just like we sent and received parameters with functions! To do this, first tell our constructor what to expect and what to do with the data:

```
User(string userName, string pass) {

    username = userName;
    password = pass;
}
```

Now if we go over to the main() we are getting an error because our class constructor is now expecting 2 strings!

```
int main(void) {

    User someUser;

    cout << "Username: " << someUser.username << "\n";
    cout << "Password: " << someUser.password << "\n";
```

Let's add those parameters and see the output:

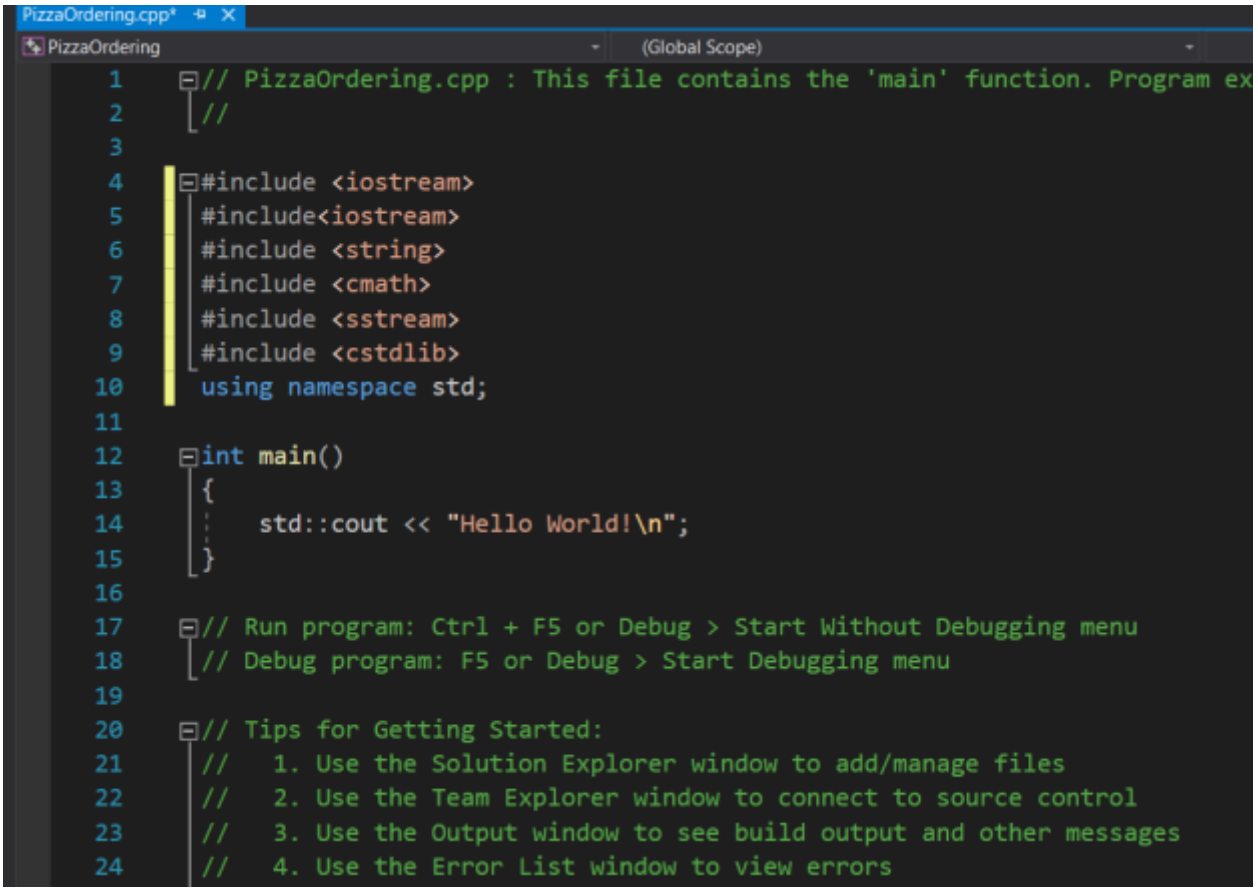
```
int main(void) {

    User someUser("Alyssa", "something");

    cout << "Username: " << someUser.username << "\n";
    cout << "Password: " << someUser.password << "\n";
```

```
Microsoft Visual Studio Debug Console
Username: Alyssa
Password: something
```

You do not need to require parameters for your constructor, nor do you have to even have a constructor - it is all up to you! Also, you can require all different types of parameters and

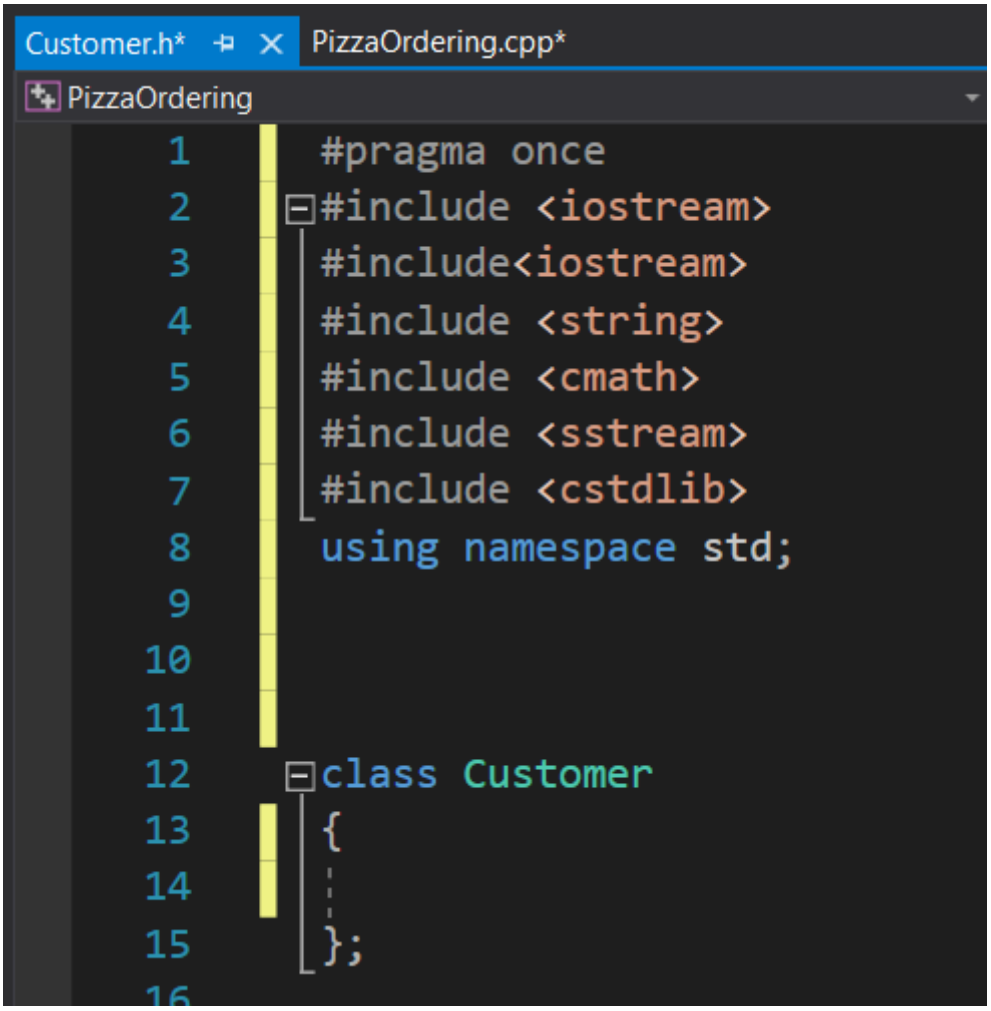
you can require 1, 3, 7, etc, whatever suits your programming needs!

Let's start an entirely new project where we will create a pizza ordering program using our own Customer class and a second Order class. Click file, new project (console application) and name your project. Then add your class and include our normal headers in all files.

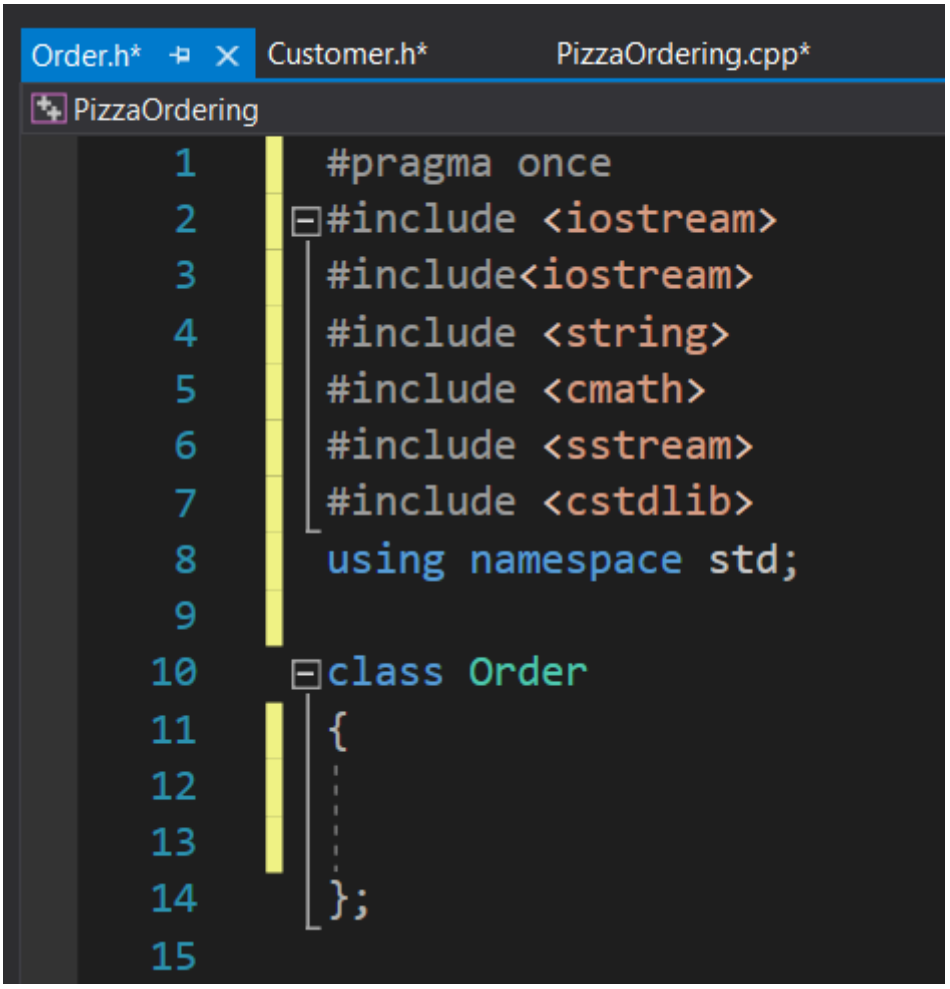


You can delete all of the green comments if you want.

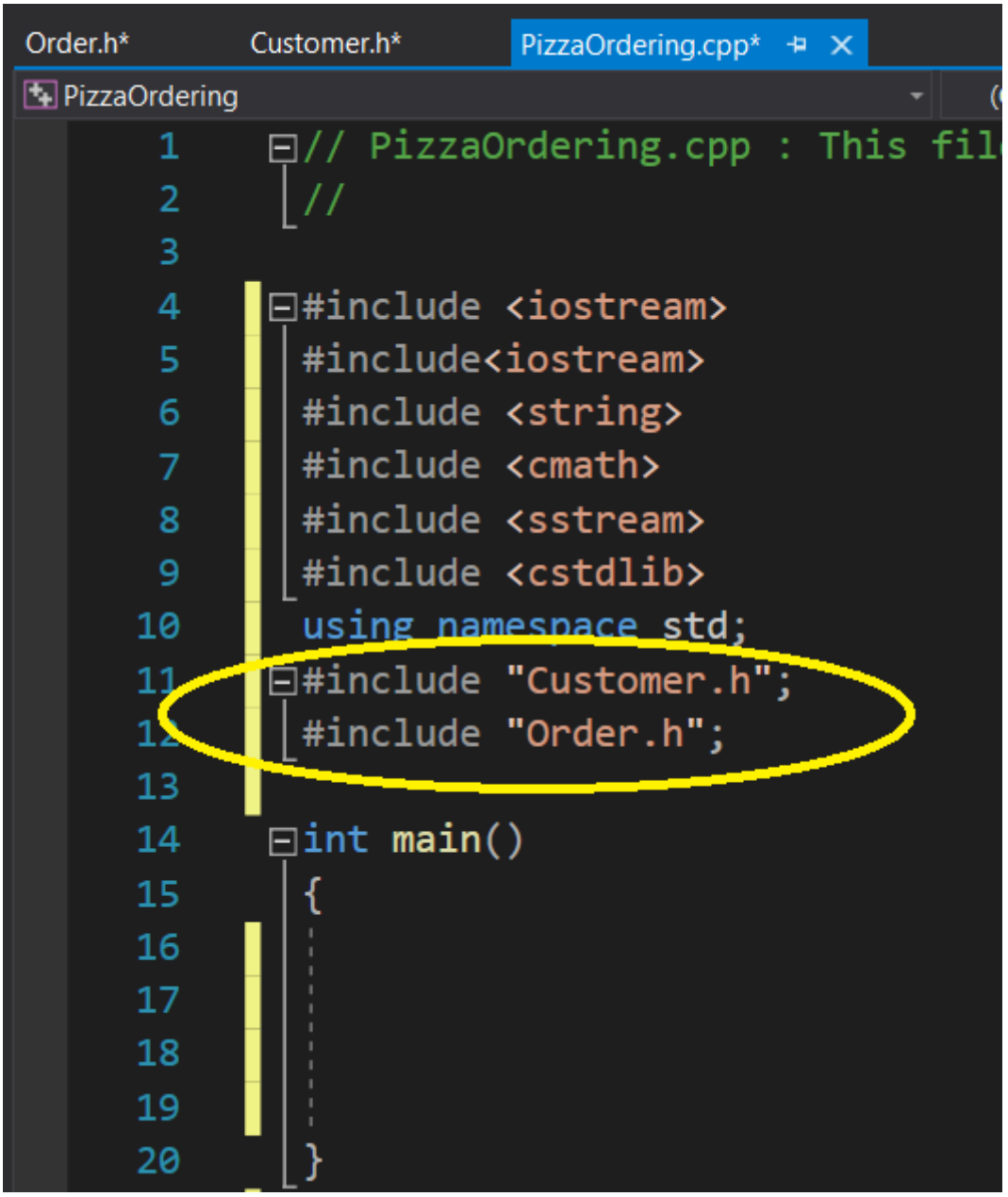
Here is our Customer class:



Here is our Order class:



Don't forget to include your Customer and Order class file in your main() .cpp file header so we can access our class!

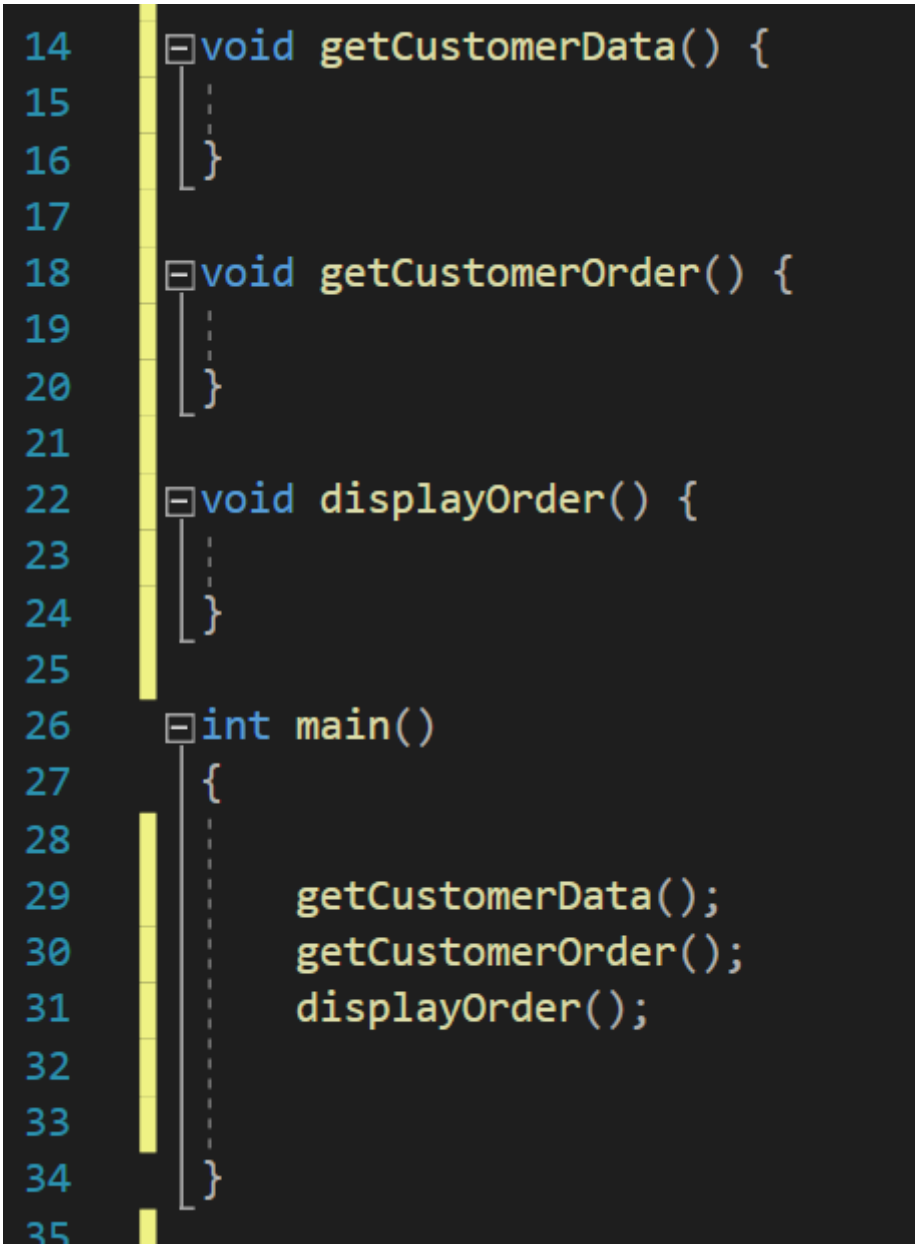


Here are the specs of what we want our program to do:

- Get the customer data including name, phone number and street address
- Get the customer order including pizza crust and 3 toppings
- Display customer order, customer data and arrival time in final output

Try it out first and see if you come up with something similar:

Let's start off in our main() by outlining the functions we want to use to get our customer data, get their order, then display the order details. We will also premake our functions:



Now, let's go over to our Customer class and set up our attributes:

```
12  class Customer
13  {
14  public:
15
16      string customerName;
17      string phoneNumber;
18      string streetAddress;
19
20  };
21
```

Then we will do the same for our Order class:

```
10  class Order
11  {
12
13  public:
14
15      string crustType;
16      string topping1;
17      string topping2;
18      string topping3;
19
20
21  };
```

Now we can go back to our main() functions and create our objects and all other programming logic. Here in our first function we have asked the user for information, then stored the data in our object's attributes to call on later. Notice how I created the class object outside of any functions and outside of the main(). This must be done so that we can call on our object when we are in any function or in the main(). If we were

to create our class object within one of these functions, the object would ***only*** be available within the function it was instantiated in!

```
10  using namespace std;
11  #include "Customer.h";
12  #include "Order.h";
13
14  Customer ourCustomer;
15
16  void getCustomerData() {
17
18      cout << "Welcome to Pizza Palace!  What is your name? ";
19      getline(cin, ourCustomer.customerName);
20
21      cout << "\n What is your phone number? ";
22      getline(cin, ourCustomer.phoneNumber);
23
24      cout << "\n What is your street address? ";
25      getline(cin, ourCustomer.streetAddress);
26
27  }
28
```

In our next function we are going to get the customer's pizza choice and store the choices in our Order class's object. Notice again where I am creating the class object:

```
14  Customer ourCustomer;
15  Order customerOrder;
16
17  void getCustomerData() { ... }
29
30  void getCustomerOrder() {
31
32      cout << "\n Thank You - Please Start Your Order!\n";
33
34      cout << "\n Choose a crust:  1 = thin crust 2 = regular crust  3 = pan crust: ";
35      getline(cin, customerOrder.crustType);
36
37      cout << "\n Choose your first topping:  1 = cheese 2 = pepperoni  3 = onion: ";
38      getline(cin, customerOrder.topping1);
39
40      cout << "\n Choose your second topping:  1 = cheese 2 = pepperoni  3 = onion: ";
41      getline(cin, customerOrder.topping2);
42
43      cout << "\n Choose your third topping:  1 = cheese 2 = pepperoni  3 = onion: ";
44      getline(cin, customerOrder.topping3);
45
46  }
```

Next, here is our displayOrder() function to call on our objects so that we can display the order information to our customer:

```
void displayOrder() {
    cout << "\nThank you for your order " << ourCustomer.customerName << "!\n";
    cout << "\nYou have ordered one " << customerOrder.crustType << " pizza with "<<customerOrder.topping1<<",
    cout << "\nYour order will arrive at " << ourCustomer.streetAddress << " within 15 minutes!\n";
    cout << "\nWe will call you at " << ourCustomer.phoneNumber << "if anything changes.";
}
```

Now we need to add some logic to our customer order because they were choosing numbers to represent different toppings and crust preferences. To do this, we need to create a method in our Order class that we can send the user choices, take those choices and set our class variable attributes with the correct data. Let's create our method and tell it what parameters we are demanding:

```
class Order
{

public:

    string crustType;
    string topping1;
    string topping2;
    string topping3;

    void setCrustAndToppings(string crust, string t1, string t2, string t3) {

    }
```

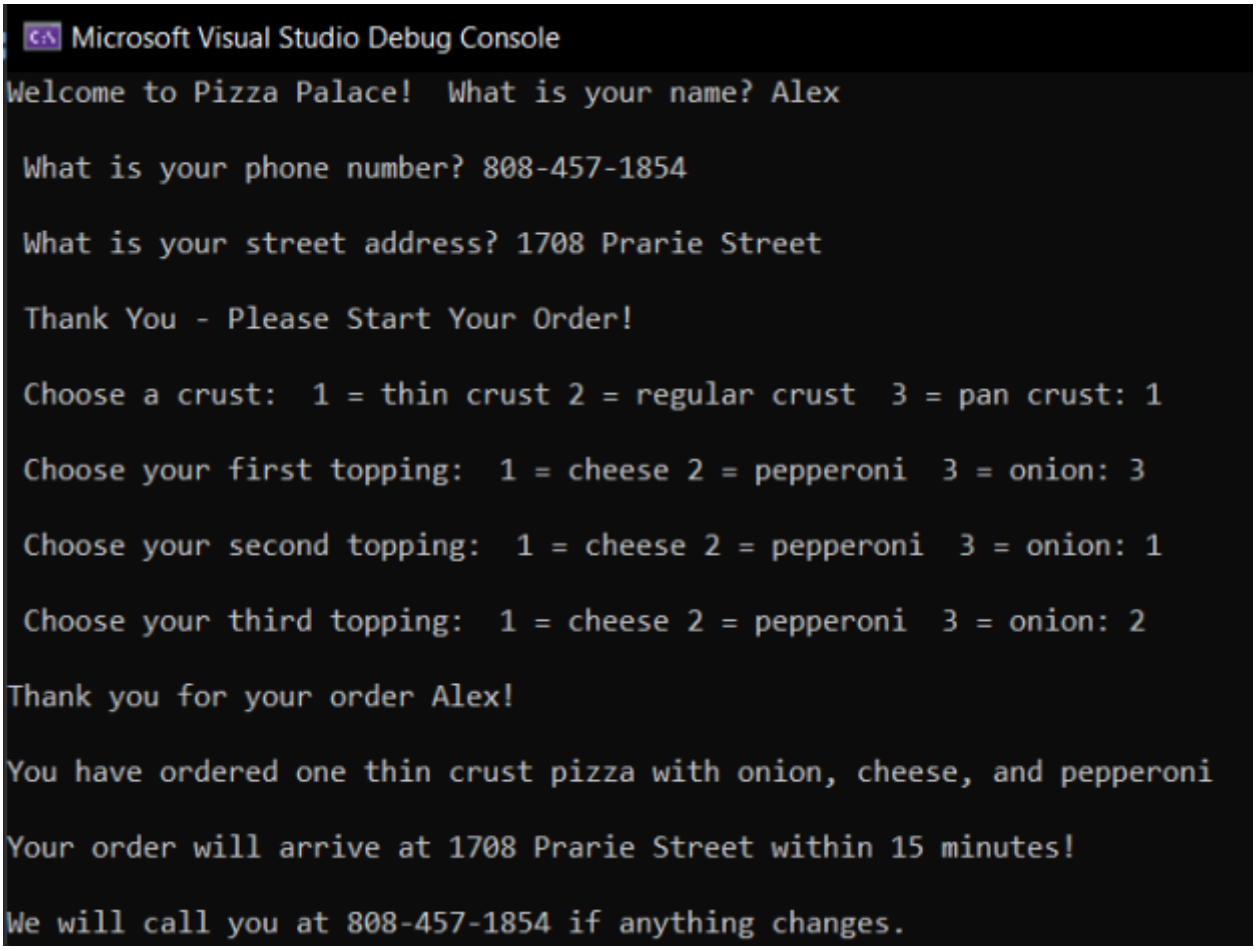
Now we can head over to the function where we are receiving the customer order and pass those choices into this object's class method.

```
void getCustomerOrder() {

    cout << "\n Thank You - Please Start Your Order!\n";

    cout << "\n Choose a crust:  1 - thin crust 2 - regular crust  3 - pan crust: ";
    getline(cin, customerOrder.crustType);

    cout << "\n Choose your first topping:  1 - cheese 2 - pepperoni  3 - onion: ";
    getline(cin, customerOrder.topping1);

    cout << "\n Choose your second topping:  1 - cheese 2 - pepperoni  3 - onion: ";
    getline(cin, customerOrder.topping2);

    cout << "\n Choose your third topping:  1 - cheese 2 - pepperoni  3 - onion: ";
    getline(cin, customerOrder.topping3);

    customerOrder.setCrustAndToppings(customerOrder.crustType, customerOrder.topping1, customerOrder.topping2, customerOrder.topping3);
}
```

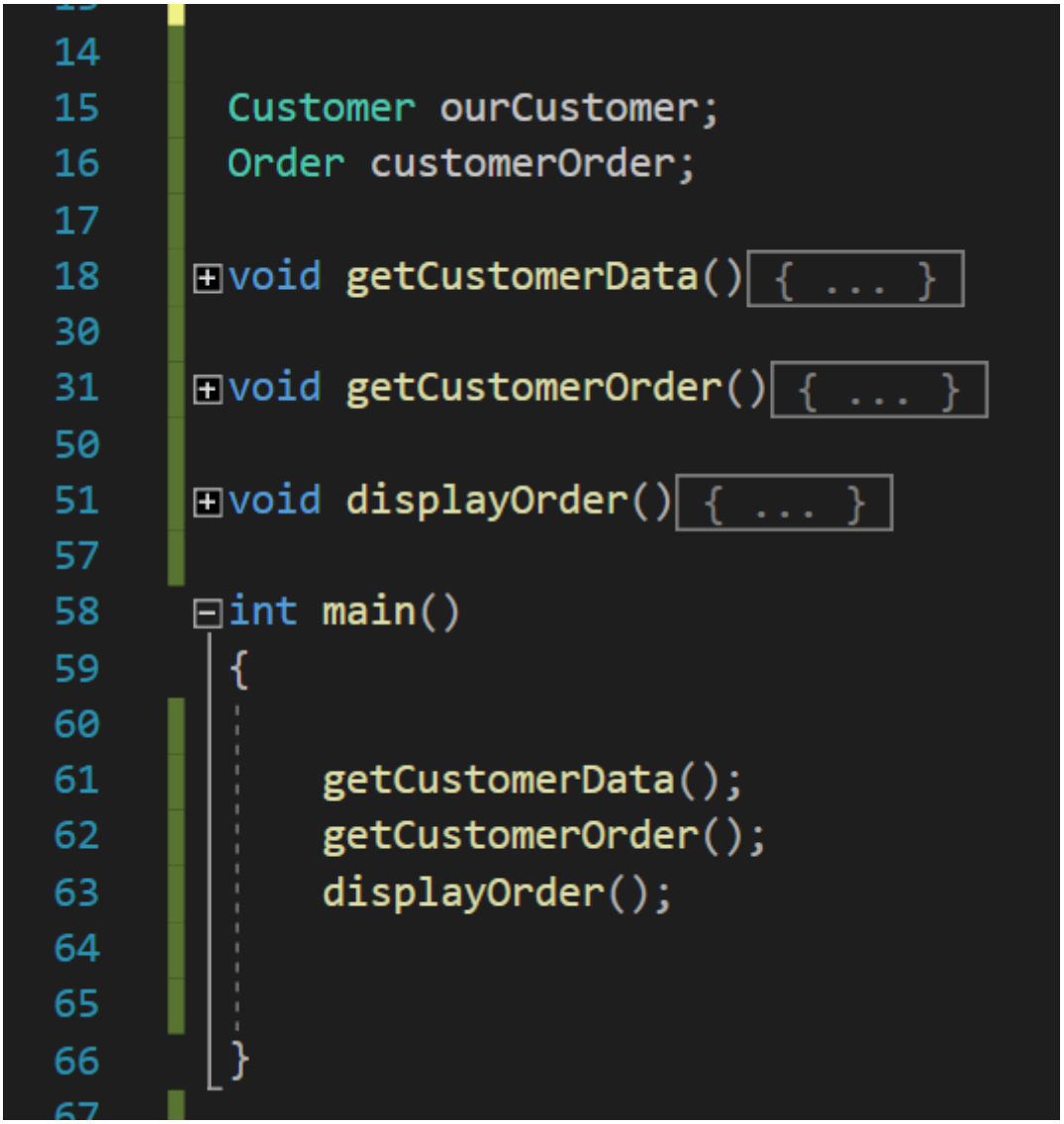
Now we can go over to our setCrustAndToppings class method to program some logic. Here is a bit of that code:

```
void setCrustAndToppings(string crust, string t1, string t2, string t3) {
    if (crust == "1") {
        crustType = "thin crust";
    }
    else if (crust == "2"){
        crustType = "regular crust";
    }
    else if (crust == "3") {
        crustType = "pan crust";
    }

    if (t1 == "1") {
        topping1 = "cheese";
    }
    else if (t1 == "2") {
        topping1 = "pepperoni";
    }
    else if (t1 == "3") {
        topping1 = "onion";
    }
```

Did you follow along and understand what is happening? Here is the output:

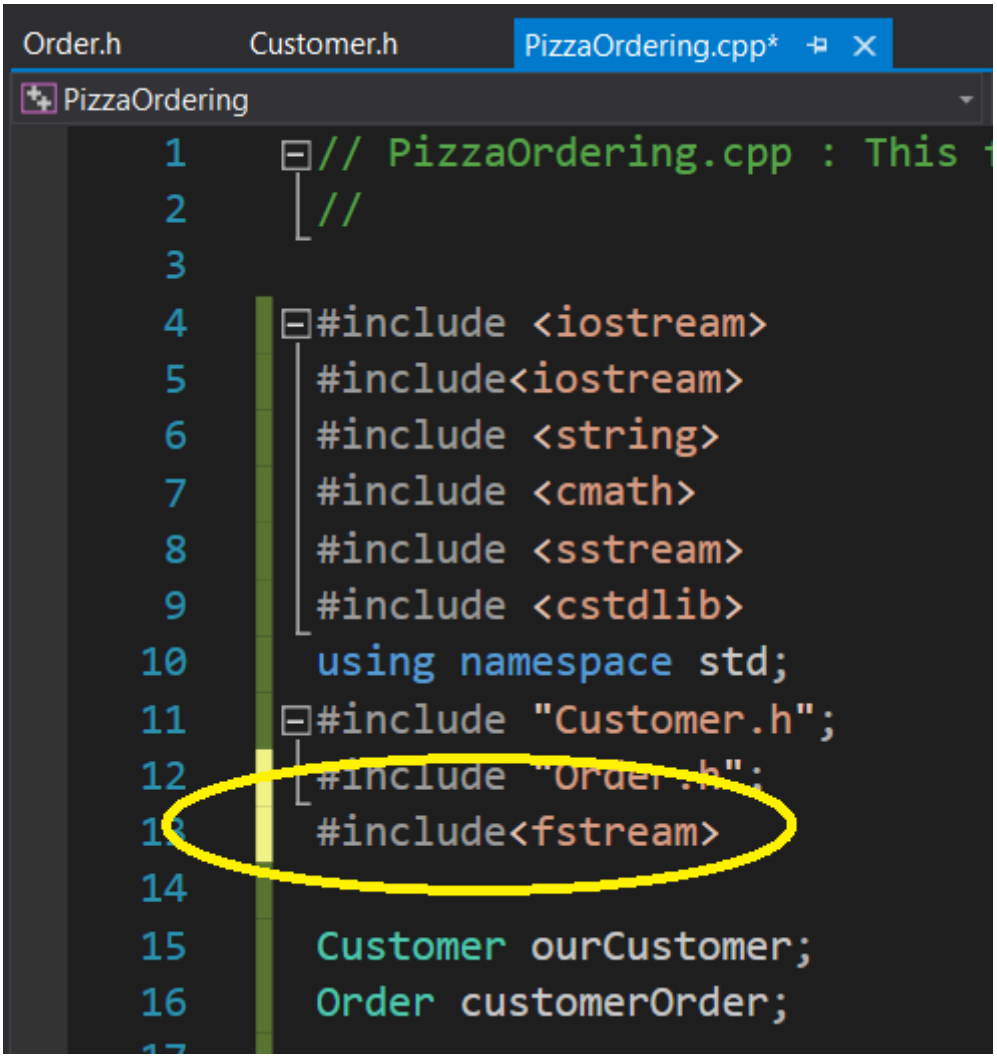

Microsoft Visual Studio Debug Console
Welcome to Pizza Palace! What is your name? Alex
What is your phone number? 808-457-1854
What is your street address? 1708 Prarie Street
Thank You - Please Start Your Order!
Choose a crust: 1 = thin crust 2 = regular crust 3 = pan crust: 1
Choose your first topping: 1 = cheese 2 = pepperoni 3 = onion: 3
Choose your second topping: 1 = cheese 2 = pepperoni 3 = onion: 1
Choose your third topping: 1 = cheese 2 = pepperoni 3 = onion: 2
Thank you for your order Alex!
You have ordered one thin crust pizza with onion, cheese, and pepperoni
Your order will arrive at 1708 Prarie Street within 15 minutes!
We will call you at 808-457-1854 if anything changes.

Good job! By using our classes and creating our own functions we have a neat and easily understandable program!

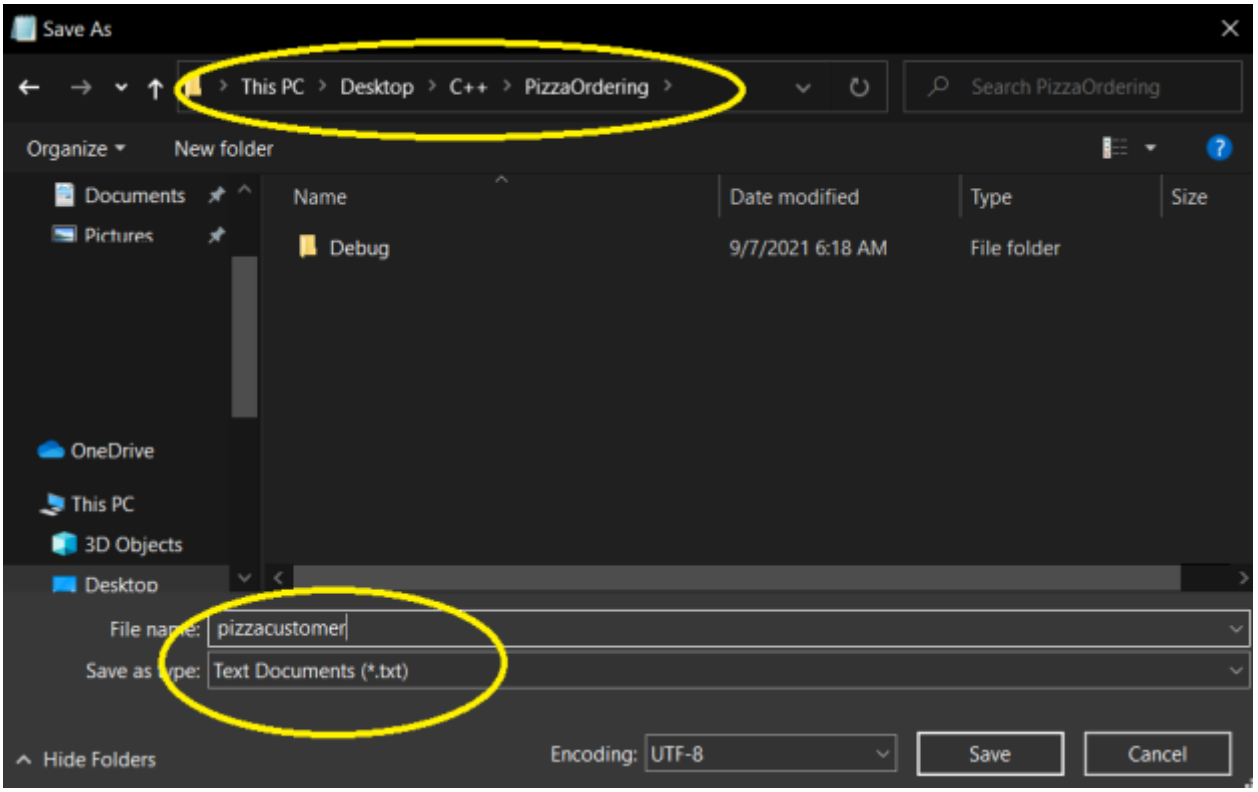

Customer ourCustomer;
Order customerOrder;
void getCustomerData() { ... }
void getCustomerOrder() { ... }
void displayOrder() { ... }
int main()
{
getCustomerData();
getCustomerOrder();
displayOrder();
}

FILES

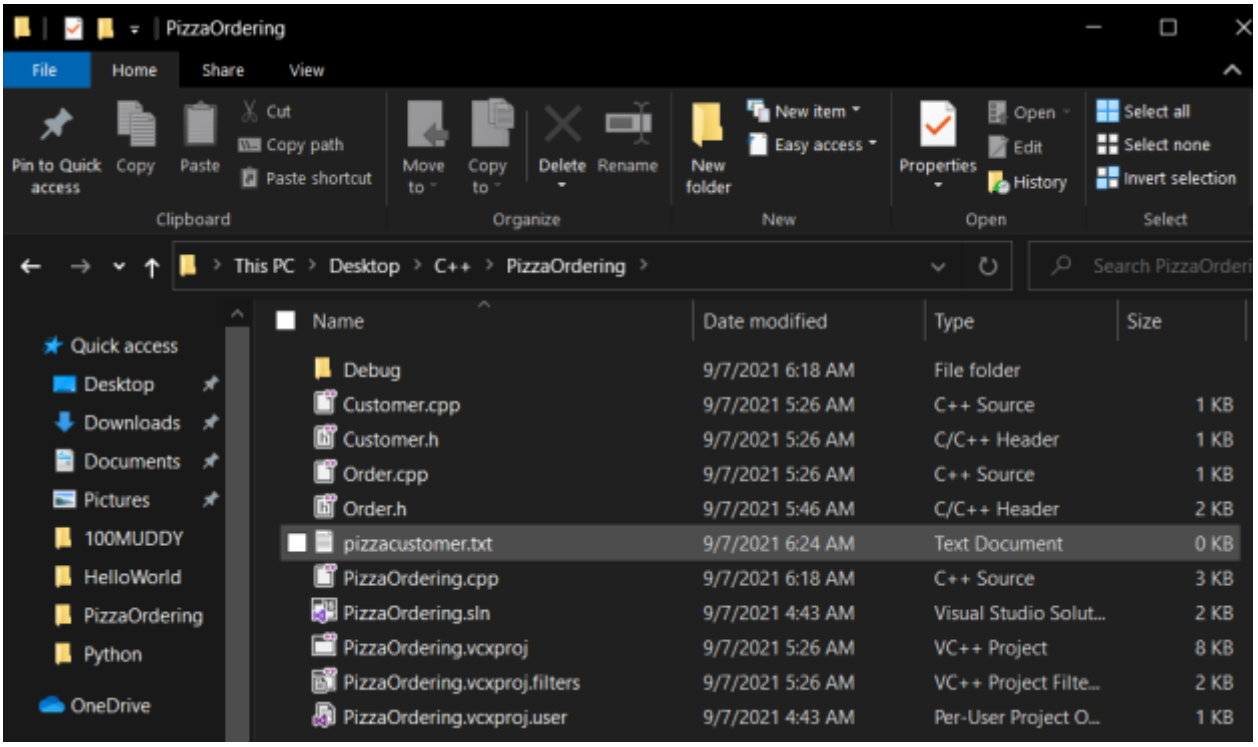
Did you know we can also read files and write to files? This is very powerful and is used a lot, think about a company's database with customer information or usernames and passwords. There are thousands of ways and reasons to read or write files. Let's use the pizza ordering app we created and write our customer's name and address to a file so we have their information saved. To start doing this we are going to include the fstream library for us to use:



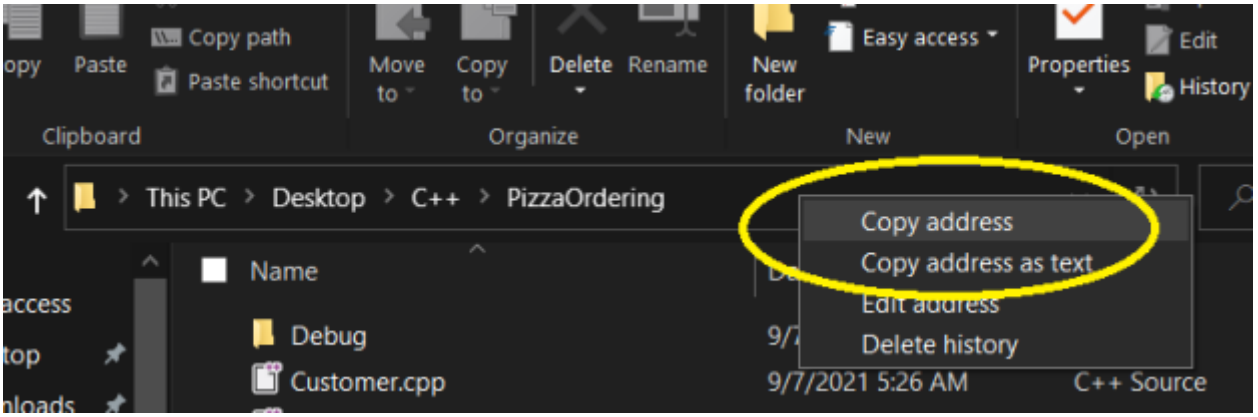
The first thing we need to do is create and open a text file. Open your notepad and save/name your .txt file in the same folder as your program:



Now, we can check that the file is there, which it is:



We need to get the path to our file so that we can tell our program where to get and open it. You can do this by right-clicking in the addess bar:



Now, you are going to place the complete file path ***and*** type in your exact file within the quotation marks as follows:

```
fstream MyFile ("C:/Users/kspri/Desktop/C++/PizzaOrdering/pizzacustomer.txt");
```

Something you have to do is replace all of the backward slashes \ with a forward slashes /. Why? Because when you place a backward slash the computer thinks you are putting in a new line (\n) or other escape characters.

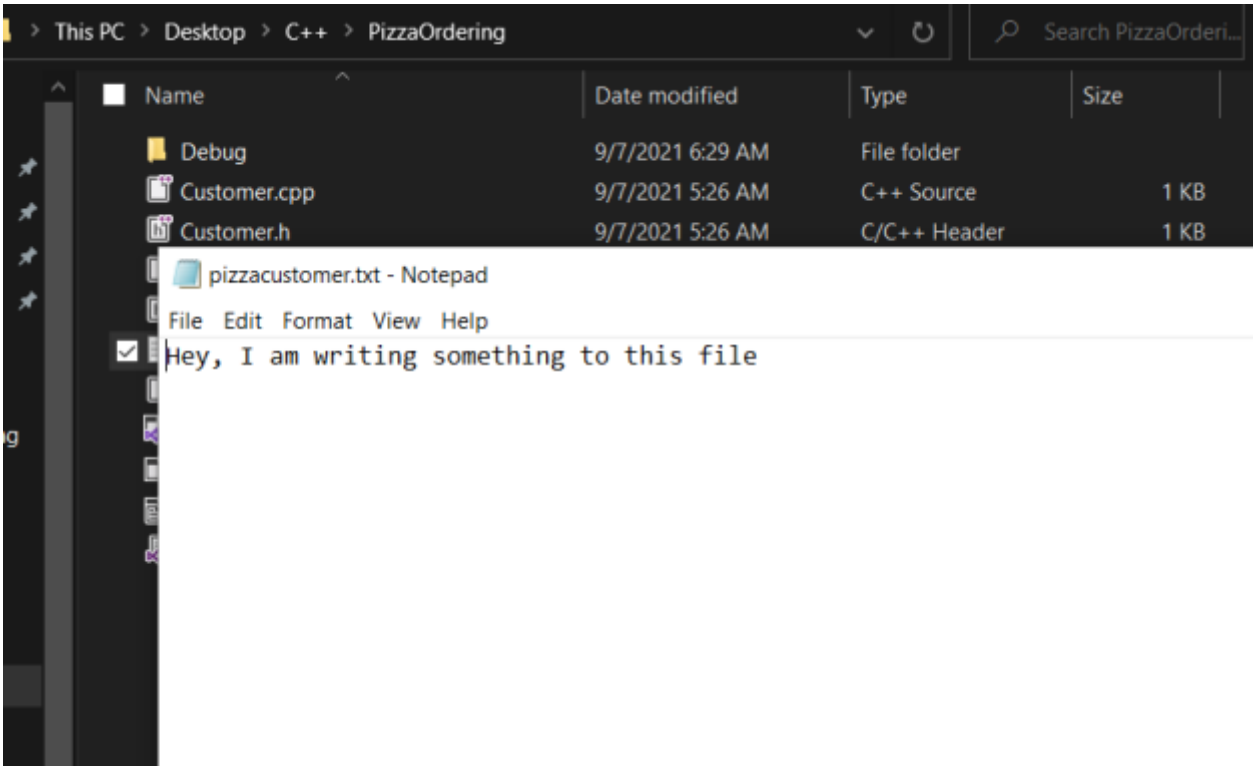
Next, we can write a string or variable to the file:

```
fstream MyFile ("C:/Users/kspri/Desktop/C++/PizzaOrdering/pizzacustomer.txt");

MyFile << "Hey, I am writing something to this file";
```

After you are done writing to a file you should always close it to clean up any unnecessary memory space... no one likes a slow app!

```
int main()
{

    getCustomerData();
    getCustomerOrder();
    displayOrder();

    fstream MyFile ("C:/Users/kspri/Desktop/C++/PizzaOrdering/pizzacustomer.txt");

    MyFile << "Hey, I am writing something to this file";

    MyFile.close();

}
```

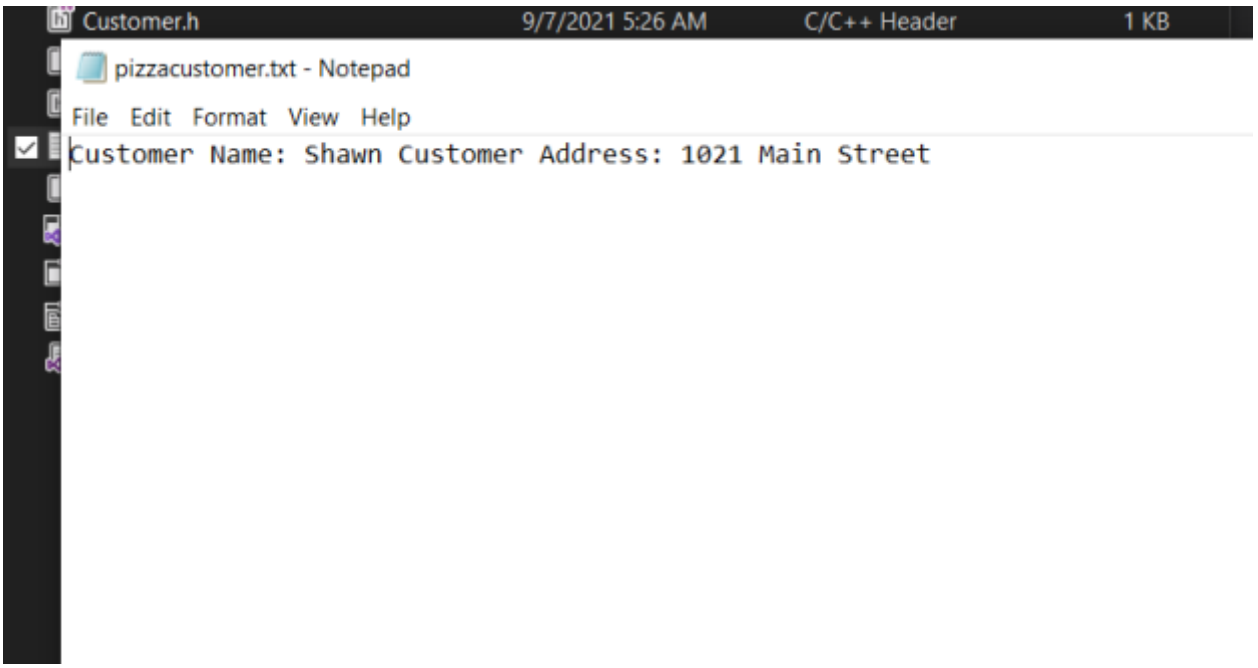
Let's run through our program, go find our file and see if it wrote to it!



Great! Now let's change what we wrote to the file and use our customer information instead:

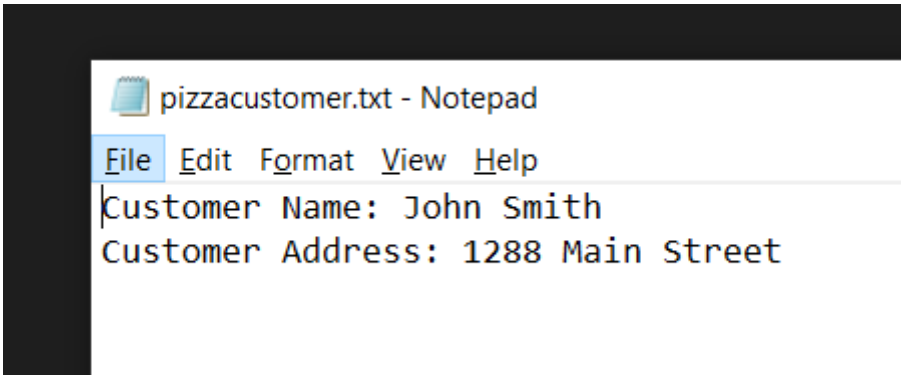
```
fstream MyFile ("C:/Users/kspri/Desktop/C++/PizzaOrdering/pizzacustomer.txt");

MyFile << "Customer Name: " <<ourCustomer.customerName<<" Customer Address: "<<ourCustomer.streetAddress;

MyFile.close();
```

Let's run through the program again and go check our file:


Customer.h 9/7/2021 5:26 AM C/C++ Header 1 KB
pizzacustomer.txt - Notepad
File Edit Format View Help
Customer Name: Shawn Customer Address: 1021 Main Street

Or, when you write to your file you could use the \n new line:

```
MyFile << "Customer Name: " <<ourCustomer.customerName<<"\nCustomer Address: "<<ourCustomer.streetAddress;
```



Pretty fun right?! Now let's read a file! To do this we we use a while loop together with the getline() function (which belongs to the ifstream class) to read the file line by line, and to print the content of the file. First, we create a string to hold the line of string text we read from the file:

```
string textFromFile;
```

Next, we will tell the computer where to read the file from using the ifstream:

```
ifstream MyReadFile("C:/Users/kspri/Desktop/C++/PizzaOrdering/pizzacustomer.txt");
```

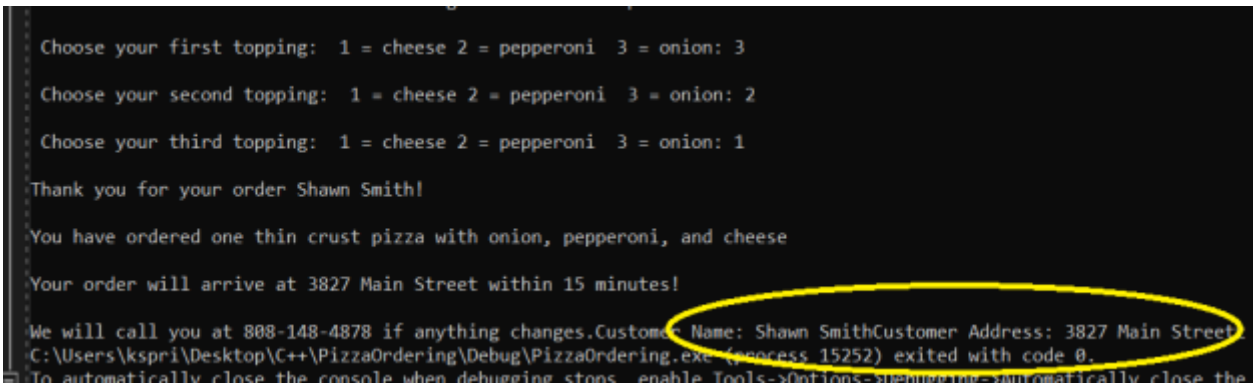
Then we will use a while loop with the getline() function to read the file line by line, and print it out line by line:

```
while (getline(MyReadFile, textFromFile)) {
    cout << textFromFile;
}
```

Finally, you will close your file:

```
64
65      fstream MyFile ("C:/Users/kspri/Desktop/C++/PizzaOrdering/pizzacustomer.txt");
66
67      MyFile << "Customer Name: " <<ourCustomer.customerName<<"\nCustomer Address: "<<our
68
69      MyFile.close();
70
71
72      string textFromFile;
73
74      ifstream MyReadFile("C:/Users/kspri/Desktop/C++/PizzaOrdering/pizzacustomer.txt");
75
76      while (getline(MyReadFile, textFromFile)) {
77          cout << textFromFile;
78      }
79
80      MyReadFile.close();
```

Let's run our program and see if the data in our file is displayed:


Choose your first topping: 1 = cheese 2 = pepperoni 3 = onion: 3
Choose your second topping: 1 = cheese 2 = pepperoni 3 = onion: 2
Choose your third topping: 1 = cheese 2 = pepperoni 3 = onion: 1
Thank you for your order Shawn Smith!
You have ordered one thin crust pizza with onion, pepperoni, and cheese
Your order will arrive at 3827 Main Street within 15 minutes!
We will call you at 808-148-4878 if anything changes.Customer Name: Shawn SmithCustomer Address: 3827 Main Street
C:\Users\kspri\Desktop\C++\PizzaOrdering\Debug\PizzaOrdering.exe (process 15252) exited with code 0.

Wow! You have come a long way and I bet you didn't even realize it!

Congrats! You deserve it! You have officially learned the crash course basics of how to code in the C++ language! Now that you have a solid foundation and understanding, there are many paths you can take. There are tons of niches within the C++ programming world, including creating full standalone games, Graphical Based Applications, Database Software, Browsers, even full Operating Systems and more! There are a lot of options as you can see! If you want to further your C++ learning I am currently writing books on each of these subjects so keep a look

out for those. I have genuinely enjoyed creating this material and I sincerely hope it has helped you. Remember, the possibilities are endless... you just have to imagine it -then code it!